AD-A016 482

# DISTRIBUTED PROCESSOR/MEMORY ARCHITECTURES DESIGN PROGRAM

G. Consolver, et al

Texas Instruments, Incorporated

Prepared for.

Air Force Avionics Laboratory

February 1975

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE

Best
Available
Copy

309076

AFAL-TR-75-90

ADA016482

# DISTRIBUTED PROCESSOR/MEMORY ARCHITECTURES DESIGN PROGRAM

*TEXAS INSTRUMENTS INCORPORATED*
*13500 NORTH CENTRAL EXPRESSWAY*
*DALLAS, TEXAS 75222*

FEBRUARY 1975

DDC
DEFENSE
OCT 22 1975
C

FINAL REPORT FOR THE PERIOD 10 DECEMBER 1973 TO 6 DECEMBER 1974

Approved for public release; distribution unlimited

AIR FORCE AVIONICS LABORATORY
Air Force Systems Command
Wright-Patterson Air Force Base, Ohio 45433

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1 REPORT NUMBER: AFAL-TR-75-80

2 GOVT ACCESSION NO

3 RECIPIENT'S CATALOG NUMBER

4 TITLE (and Subtitle): Distributed Processor Memory Architectures Design Program

5 TYPE OF REPORT & PERIOD COVERED: Final Report 10 Dec 73 - 7 Dec 74

6 PERFORMING ORG. REPORT NUMBER

7 AUTHOR(s): Mr George Consolver, et al

8 CONTRACT OR GRANT NUMBER(s): F33615-74-C-1018

9 PERFORMING ORGANIZATION NAME AND ADDRESS: Texas Instruments Incorporated, 13500 North Central Expressway, Dallas, Texas 75222

10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 2003 04 04

11 CONTROLLING OFFICE NAME AND ADDRESS: Air Force Avionics Laboratory, Air Force Systems Command, Wright-Patterson AFB, Ohio 45433

12 REPORT DATE: February 75

13 NUMBER OF PAGES: ~~XXX~~ 500

14 MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15 SECURITY CLASS. (of this report): UNCLASSIFIED

15a DECLASSIFICATION/DOWNGRADING SCHEDULE

16 DISTRIBUTION STATEMENT (of this Report): Approved for Public Release, Information Unlimited

17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18 SUPPLEMENTARY NOTES

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Processor
Computer
Avionics
Computer Simulation
Avionics Software
Microprocessor
Multiplexing

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

The purpose of Distributed Processor/Memory Architectures (DP/M) design program was to extend the DP/M avionic system processing concept to a detailed system hardware and software design. The functional design for the DP/M Processing Element (PE) is summarized, including the processor, memory, input/output interface, and a dual-level time-division-multiplex bus interface unit. A set of simulation and analysis programs was developed for modeling both the high-level network interaction among interconnected Processing Elements and the detailed internal operation of the PE. Other major areas examined were the executive control software, the process construction methodology required to develop and allocate real-time software for DP/M, and methods that could be used with DP/M to promote avionic system fault tolerance.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Best Available Copy

## FOREWORD

This final technical report presents the results of the Distributed Processor/Memory Architectures (DP/M) design program. The report addresses specific tasks performed to meet the requirements of Air Force Contract No. F33615-74-C-1018 for the Air Force Avionics Laboratory. The report was written and research performed under the direction of Mr. M. Moore, AFAL/AAM.

This report, and the engineering data submitted with it, contains the results of the DP/M design program conducted by Texas Instruments Incorporated.

Principal contributors to this report are Mr. G. Consolver, Mr. D. Ackley, Mr. M. Rickard, Mr. R. McAfee, Mr. T. Shipchandler, and Dr. V. Gylys.

## TABLE OF CONTENTS


| Section | Title | Page |
|---|---|---|
| I | INTRODUCTION AND SUMMARY | 13 |
| | A. DPM System Concept and Overview | 13 |
| | B. Summary of Design Results | 15 |
| | 1. DPM System Network Design | 15 |
| | 2. DPM Functional Simulation System | 15 |
| | 3. PE Processor, Memory, and Internal Bus Structure | 16 |
| | 4. PE Local and Global Bus Interface Unit | 17 |
| | 5. PE I/O Interface Unit | 17 |
| | 6. PE Technology and Cost Projection | 17 |
| | 7. Executive Software | 18 |
| | 8. Process Construction Study | 19 |
| | 9. Fault Tolerance Analysis Activities | 20 |
| | C. Report Overview | 20 |
| II | SYSTEM DESIGN CONSIDERATIONS | 21 |
| | A. Design Concepts | 21 |
| | 1. All-Semiconductor Design Concept | 21 |
| | 2. Architecture Selection Concept | 21 |
| | 3. Compatibility Concept | 21 |
| | 4. Usability Concept | 23 |
| | 5. Modularity Concept | 24 |
| | B. System Applications | 26 |
| | 1. Stand Alone Computation Unit | 26 |
| | 2. Smart Sensor or Standard Interface Unit | 26 |
| | 3. Distributed Function Configuration | 28 |
| | C. Overview of System Communication Interconnection Facilities Design | 29 |
| III | INFORMATION TRANSFER BUSES | 33 |
| | A. Overview | 33 |
| | B. System Requirements and Considerations | 34 |
| | 1. Distributed Systems Environment | 34 |
| | 2. Aircraft Equipment Compatibility | 34 |
| | 3. System Performance Requirements | 35 |
| | 4. System-Level Fault-Tolerance Implications | 39 |
| | C. Busing Facility Design Characteristics and Tradeoff Considerations | 39 |
| | 1. Data Link Type | 39 |
| | 2. Bus Language | 40 |
| | 3. Bus Control | 40 |
| | 4. Bus Assignment | 42 |
| | 5. Bus Communication Method | 43 |
| | 6. Bus Synchronization | 45 |
| | 7. Message Structure | 46 |
| | 8. Message Reception Control | 51 |
| | 9. Fault-Tolerance Considerations | 54 |


Preceding page blank

D. Bus Interface Unit Functional Design Description . . . . . 62
1. Bus Data Translation . . . . . 62
2. Message Reception Control . . . . . 64
3. Message Transmission Control . . . . . 69
4. Bus Access Control . . . . . 69
5. Processor Memory Interface Control . . . . . 72
6. Redundant Bus Management . . . . . 72
E. Bus Interface Unit Implementation Considerations . . . . . 72
1. BIU Register Level Description . . . . . 74
2. Hardware Complexity Estimates . . . . . 94
F. BIU Simulation Modeling . . . . . 94
G. Areas of Further Investigation . . . . . 107

IV PROCESSOR/MEMORY DESIGN . . . . . 109

A. Processor Functional Architecture . . . . . 109
Addressable Registers . . . . . 110
2. Processor Word Length . . . . . 112
3. Data Types . . . . . 113
4. Instruction Complement Selection . . . . . 113
5. Instruction Usage Observations . . . . . 123
6. Processor Interrupt Structure . . . . . 127
7. Processor Initialization . . . . . 131
8. Processor Functional Design Summary . . . . . 132
B. Processor Physical Design . . . . . 134
1. PI Instruction Cycle . . . . . 136
2. Processor Hardware Design . . . . . 139
3. Processor Complexity Estimate . . . . . 149
C. PI Memory Specification . . . . . 153
1. Memory Protection . . . . . 154
2. Error Detection . . . . . 155
3. Allocation of Memory Space . . . . . 155
D. PI Internal Interconnection . . . . . 156
1. Data Transfer and Control . . . . . 159
2. I-Bus Master Control . . . . . 159
3. I-Bus Interrupt Control . . . . . 163
4. General I-Bus Facilities and Special Functions . . . . . 163
E. Maintenance Design Considerations . . . . . 167
1. PI Hardware Checkout . . . . . 167
2. PI Built-In-Test (BIT) Concept . . . . . 167
3. Programmer Maintenance Panel Concept . . . . . 169
F. Processor Memory Design Summary . . . . . 170

V EXTERNAL DEVICE COMMUNICATIONS INTERFACE . . . . . 171

A. System Input Output Approach . . . . . 171
B. Input Output Interface Functional Requirements . . . . . 173
1. Standardized Digital Device Interface Compatibility . . . . . 173
2. Data Rate Requirements . . . . . 173
3. Program Interference . . . . . 174
4. PI Support Functions . . . . . 174
5. Implementation Considerations . . . . . 174
6. Extend Capability Considerations . . . . . 175

C. IOU Functional Design Description . . . 175
1. Standardized Data Transfer Channel . . . 175
2. Programmable Interval Timer . . . 177
3. PE Initialization Control . . . 177
4. PE Clock Control . . . 190
5. Processor Memory Interface Control . . . 190
D. Hardware Complexity Estimates . . . 192

VI PE IMPLEMENTATION CONSIDERATIONS . . . 193

A. Introduction . . . 193
B. Technology . . . 194
1. Memory Technology . . . 195
2. Logic Technology . . . 199
3. Summary of Technology Recommendations . . . 203
C. PE Performance Projections . . . 203
D. PE Partitioning and Packaging . . . 206
1. Processor . . . 207
2. Memory and Control . . . 207
3. Bus Interface Unit . . . 210
4. Input/Output Interface Unit . . . 220
E. Cost Projections . . . 233
1. Commercial Micro Computer . . . 233
2. Commercial Grade DP/M PE . . . 234
3. Military Grade DP/M PE . . . 234
4. Cost Conclusion . . . 234

VII FUNCTIONAL SIMULATION SYSTEM . . . 235

A. Section Overview . . . 235
B. Introduction . . . 235
C. Simulation Approach . . . 236
1. Definition of Terms . . . 236
2. Simulation Control Structure . . . 236
3. Basic Simulator . . . 237
D. System Network Simulator . . . 240
1. System Network Simulator Models . . . 240
2. System Network Simulator List Structures . . . 240
3. Terminal List Structure . . . 241
4. PE Interconnectivity . . . 242
5. Logical Precedence Relationships Between Tasks . . . 243
6. DP/M Test Mission Characteristics . . . 243
7. Data Analysis and Recording Example . . . 247
8. Virtual Message Distributor . . . 250
9. Model Dependent Routines . . . 253
10. Bus Control Model Events . . . 257
11. Avionic Task Model Events . . . 257
12. Executive Model Events . . . 257
13. Data Collection and Report Generation . . . 259
E. Processing Element Simulation . . . 261
1. Processing Element Simulator Organization . . . 269
2. Processor Simulation . . . 270
3. Bus Interface Unit Simulation . . . 277
4. I/O Simulation . . . 287

5. Memory Simulation . . . . . 290
6. Ancillary Routines . . . . . 290
7. Summary . . . . . 293

VIII EXECUTIVE DESIGN AND SYSTEM OPERATION . . . . . 295

A. Executive Design Considerations . . . . . 295
B. Executive Control Overview . . . . . 298
C. Executive Structure Hierarchy . . . . . 300
D. Functional Executive Design Overview . . . . . 302
1. LEX Functional Design . . . . . 302
2. GEX Functional Design . . . . . 304
E. Local Executive Detailed Design and Operation . . . . . 304
1. Application Task LEX Tables . . . . . 305
2. LEX Usage of Hardware Resources . . . . . 306
3. Detailed LEX Module Description . . . . . 311
F. Detailed GEX Program Description . . . . . 325
1. Global Executive Data Base and Hardware Usage . . . . . 325
2. Scheduler (GEXSCHED) . . . . . 330
3. Clock Update Task . . . . . 336
4. LEX Routines in GEX . . . . . 336
5. Message Output Module . . . . . 336
6. Activate/Deactivate Monitor . . . . . 336
7. Completion Status Monitor . . . . . 337
8. Mode Change Detector . . . . . 337
G. Sample Illustration of LEX, GEX Operation . . . . . 338
H. System Operation Considerations/Observations . . . . . 341
I. Simulation Model Development . . . . . 343
J. System Initialization Procedure . . . . . 345
1. Initial State of the System . . . . . 346
2. ROM Bootstrap Program . . . . . 346
3. Software Bootstrap Program Load . . . . . 347
4. System Program Loads . . . . . 347
5. System Synchronization . . . . . 348
K. Summary of DP M Executive Structure . . . . . 349

IX PROCESS CONSTRUCTION STUDY . . . . . 351

A. Introduction . . . . . 351
1. Process Construction Problem Statement . . . . . 351
2. Overview . . . . . 353
B. Problem Analysis . . . . . 354
1. Basic Concepts and Definitions . . . . . 354
2. Process Development Methodology Based on Top-Down Modeling and Successive Simulations . . . . . 362
3. Scope and Automation of Process Construction . . . . . 373
4. Recommended Approach for the Development of Process Construction Capabilities . . . . . 378
5. Analysis of Process Construction Algorithms . . . . . 381
C. Summary of Requirements . . . . . 400
1. Assumptions . . . . . 401
2. Interaction With Other Phases of Process Development and With Other Systems . . . . . 403
3. Programming Languages . . . . . 404

4. Process Development Information System . . . . . 406
5. Functional Specification of the Process Construction Procedure . . . . 410
6. Process Construction Algorithms and Input/Output Data Sets . . . . . 419
7. Implementational Requirements . . . . . 422
D. Process Construction Case Study . . . . . 423
1. Mission Segment Description Summary . . . . . 424
2. Mission Software Models . . . . . 424
3. Computer Definition Model . . . . . 425
4. Application Program Timing and Sizing Effort . . . . . 425
5. Partitioning Results . . . . . 430
6. Simulation Description . . . . . 432
7. Simulation Results . . . . . 435
E. Recommended Future Activity in DP/M Process Construction . . . . . 435
1. Process Construction Inputs . . . . . 436
2. Process Construction Outputs . . . . . 439

X FAULT TOLERANCE ANALYSIS . . . . . 441

A. Introduction . . . . . 441
B. Fault Tolerance Approach . . . . . 441
1. Functional Redundancy . . . . . 442
2. Physical Redundancy . . . . . 443
C. Approach to Fault Spotting . . . . . 444
D. Failure Recovery Approach . . . . . 446
E. Power Supply Considerations . . . . . 450
F. Summary and Conclusions . . . . . 460

BIBLIOGRAPHY . . . . . 497

APPENDIXES

A Basic Process Construction Input/Output Data Files . . . . . 463
B Process Construction Case Study Simulation Report . . . . . 483

LIST OF ABBREVIATIONS . . . . . 499

## LIST OF ILLUSTRATIONS

| *Figure* | *Title* | *Page* |
|---|---|---|
| 1 | DP/M System Architecture | 14 |
| 2 | U.S. Electronics Industry Trends | 23 |
| 3 | Modularity Learning Curve Effects | 25 |
| 4 | Example Data Input | 27 |
| 5 | General DP/M System Network Configuration | 29 |
| 6 | System Interconnection and Communication Structure (Example) | 30 |
| 7 | Worst Case Timing Analysis of PE Bus Message Servicing | 38 |
| 8 | Manchester II Data/Synchronization Single Waveforms | 41 |
| 9 | Inter-PE Bus Message Structure/Format | 50 |
| 10 | Quasi-Global Bus Configuration | 59 |
| 11 | Redundant Global Bus Configuration | 59 |
| 12 | Global Bus Redundancy Management Functional Approach | 61 |
| 13 | BIU Functional Block Diagram | 63 |
| 14 | Message Identity Associative Addressing Operation | 65 |
| 15 | Message Reception Control Initialization Procedure | 66 |

16 BIU Memory Allocation Map . . . . . 68
17 Message Transmission Initialization Procedure . . . . . 70
18 Bus Control Sequence (Supercommutation Example) . . . . . 71
19 Bus Interface Translation Register-Level Block Diagram . . . . . 74
20 Message Reception Control Register-Level Block Diagram . . . . . 75
21 Message Reception Controller Flow Diagram . . . . . 76
22 Message Transmission Control Register-Level Block Diagram . . . . . 82
23 Message Transmission Controller Flow Diagram . . . . . 83
24 Bus Access Control Register-Level Block Diagram . . . . . 87
25 Bus Access Controller Flow Diagram . . . . . 88
26 Processor/Memory Interface Control Register-Level Block Diagram . . . . . 92
27 Redundant Bus Management Register-Level Block Diagram . . . . . 93
28 Global Bus Interface Register-Level Block Diagram . . . . . 95
29 Local Bus Interface Register-Level Block Diagram . . . . . 99
30 Global/Local Message Output Event Scheduling Flow . . . . . 104
31 Global/Local Message Input Event Scheduling Flow . . . . . 106
32 Byte Data Representation . . . . . 113
33 Byte Usage . . . . . 113
34 Multiple Precision Data Representation . . . . . 114
35 Standard Instruction Format . . . . . 118
36 Extended Short Instruction Formats . . . . . 119
37 Memory Input/Output (Special Extended Short) Format . . . . . 121
38 DP/M Processor Instruction Formats . . . . . 123
39 DP/M Stack Operations . . . . . 127
40 Sample FORTRAN Subroutine Linkage . . . . . 128
41 Status Word Format . . . . . 129
42 Restoring of Program Counter and Status Word . . . . . 131
43 Restoring of Program Counter and Status Word . . . . . 132
44 Basic Instruction Cycle With Interrupts . . . . . 134
45 Processor Block Diagram . . . . . 135
46 Basic Instruction Cycle . . . . . 137
47 DP/M Instruction Cycle . . . . . 138
48 Direct Indexed Operand Derivation . . . . . 140
49 Load Instruction Execution . . . . . 141
50 Add Instruction Execution . . . . . 142
51 Load Direct Short . . . . . 143
52 Load Constant Short . . . . . 144
53 Load Register . . . . . 145
54 Instruction Register, Select and Decoder Diagram . . . . . 146
55 Processor Control Diagram . . . . . 146
56 Micro Sequencer Control (μSC) . . . . . 147
57 Micro Data Processor . . . . . 148
58 Status Register and Control . . . . . 148
59 PLA Versus ROM Layout . . . . . 150
60 PE Memory Allocation . . . . . 155
61 DP/M PE Subelement . . . . . 156
62 DP/M Modules . . . . . 158
63 I-Bus Master to Slave (Memory-I/O) Send Cycle (Delay is Exaggerated for Clarity) . . . . . 160
64 I-Bus Master From Slave (Memory-I/O) Receive Cycle (Delay is Exaggerated for Clarity) . . . . . 161
65 I-Bus Master Device Interconnection . . . . . 162

66 I-Bus Master Bus Request Cycle . . . 164
67 Interrupt Interface Sequence . . . 165
68 Processor Initialization (Clear/Reset) Sequence . . . 166
69 Programmer's Maintenance Panel . . . 168
70 Input/Output Data Interface and System Distribution . . . 172
71 Input/Output Interface Unit Functional Block Diagram . . . 176
72 DP/M PE Input/Output Interface Signals . . . 178
73 I/O Data Transfer Channel Register-Level Block Diagram . . . 179
74 Autonomous I/O Data Transfer Controller Flow Diagram . . . 181
75 Autonomous Data Channel Transfer Control Word Format . . . 189
76 PE Initialization (Clear/Reset) Control Signal Sequence . . . 191
77 DP/M Block Diagram . . . 193
78 Performance and Applications Versus Technology . . . 200
79 Complementary MOS (CMOS) . . . 201
80 Integrated Injection Logic $I^2L$ . . . 202
81 Extended Short Instructions . . . 204
82 Standard Format Instructions . . . 205
83 DP/M Processor . . . 208
84 DP/M Memory Controller . . . 209
85 DP/M Memory . . . 210
86 Multi-Device BIU Partitioning . . . 213
87 Bus Interface Logic Unit (BILU) Device Attributes . . . 215
88 Bus Interface Translation Unit (BITU) Device Attributes . . . 216
89 Redundant Bus Management Unit (RBMU) Device Attributes . . . 217
90 Redundant Bus Management Unit (RBMU) Device Implementation Alternative . . . 218
91 Bus Interface Unit (BIU) Single-Device Attributes . . . 219
92 Input/Output Logic Unit (IOLU) Device Attributes . . . 221
93 Input/Output Interface Unit Physical Partitioning . . . 222
94 Input/Output Interface Unit Modular Device Family Partitioning . . . 224
95 Autonomous Transfer Controller Logic (ATCL) Device Attributes . . . 225
96 Autonomous Transfer Controller Registers (ATCR) Device Attributes . . . 226
97 Input/Output Data Interface (IODI) Device Attributes . . . 227
98 Bus Master Interface (BMI) Device Attributes . . . 228
99 Bus Slave Interface (BSI) Device Attributes . . . 229
100 Bus Slave Interface (BSI) Device Attributes . . . 230
101 Programmable Interval Timer (PIT) Device Attributes . . . 232
102 Simulation Control Structure . . . 237
103 Terminal List Structure . . . 241
104 PE Interconnectivity List Structure . . . 242
105 DP/M Test Mission Time Line . . . 246
106 Directed Graph Representation Convention . . . 249
107 Loran-Directed Graph . . . 250
108 Task List Structure . . . 252
109 Physical Assignment Report . . . 254
110 Event-Level Report Format . . . 259
111 Sample Period Bus Report . . . 260
112 Sample Period Processor Report . . . 262
113 Bus Decomposition Report . . . 263
114 Bus Loading Bar Graph . . . 264
115 Bus Usage Summary Report . . . 265
116 Message Transmission Summary Report . . . 266
117 Processor Usage Bar Graph . . . 267

118 Processor Usage Summary Report . . . . . 268
119 PE Block Diagram . . . . . 269
120 PE Event Sequencing . . . . . 270
121 Instruction Execution Cycle Block Diagram . . . . . 270
122 Instruction Format Types . . . . . 272
123 Interrupt Event Notice List Structure . . . . . 275
124 Interrupt Stimulus Generation Functional Flow Chart . . . . . 276
125 Example of Instruction-Level Simulator Output . . . . . 278
126 Simulation Instruction Usage Histogram . . . . . 279
127 BIU Functional Model/PE Functional Simulation Interface . . . . . 281
128 BIU Input Message Data Example . . . . . 281
129 BIU Simulation Event Notice List Structure . . . . . 282
130 Data Input End Event Model . . . . . 284
131 Data Input End Event Routine . . . . . 285
132 Example of Bus Interface Unit Simulation Output . . . . . 286
133 System Control Flow Diagram . . . . . 291
134 Assembly Listing Example . . . . . 292
135 Static Memory Usage Report . . . . . 293
136 Example of a Directed Graph . . . . . 297
137 Directed Graph Representation of a Subfunction . . . . . 298
138 GEX LEX Scheduling Philosophy . . . . . 300
139 Executive Control Hierarchy . . . . . 301
140 LEX Module and Task Interrelationship . . . . . 303
141 GEX Block Diagram . . . . . 305
142 Task Preamble Tables Used by LEX . . . . . 307
143 Software Input Message Correlation Scheme . . . . . 310
144 Task Scheduler Subroutine Interrelationships . . . . . 312
145 Task Scheduler Message Scanner . . . . . 315
146 Task Scheduler Service Module . . . . . 321
147 Subfunction Preamble Tables Used by GEX . . . . . 226
148 Time-Ordered Linked List for GEX Scheduler . . . . . 327
149 GEX Scheduler Flow Chart . . . . . 331
150 Hypothetical DP/M System . . . . . 338
151 Hypothetical Directed Graph of Subfunction In Affinity Group 2 (AG2) . . . . . 339
152 Time Line Description of Illustrative Example . . . . . 340
153 GEX Scheduler Timing Equations . . . . . 345
154 Hardware Bootstrap Procedure . . . . . 347
155 Software Bootstrap Procedure Code . . . . . 348
156 Data Flow Graph . . . . . 360
157 Control Flow Graph . . . . . 361
158 Relationship Between the Maximally-Parallel Program Graphs . . . . . 363
159 Evolution of Process Model . . . . . 367
160 Overall Structure of the Relationship Between System Network, Functional and Instructional Level Simulators . . . . . 368
161 Use of Simulation In The Top-Down System Design Process . . . . . 370
162 Relationship Between the Process, Its Environment, and Its Response to the Environment . . . . . 371
163 Process Construction of Moderate Scope . . . . . 375
164 Program Analysis Report, Page 1 . . . . . 388
165 Program Analysis Report, Page 2 . . . . . 389
166 Program Analysis Report, Page 3 . . . . . 390

167 Program Analysis Report, Page 4 . . . 391
168 Program Analysis Report, Page 5 . . . 392
169 Program Model Report, Page 1 . . . 393
170 Program Model Report, Page 2 . . . 394
171 Block Diagram of Process Construction Procedure . . . 413
172 Program Synthetic Model Summary Report . . . 425
173 Task Synthetic Model Summary Report . . . 426
174 Computer Definition Model . . . 429
175 INSNAV Sizing Report . . . 430
176 Program Graph of INSNAV (Inertial Navigation) . . . 432
177 Case Study System Control and Message Flow . . . 433
178 Automation of the Initial and Final Stages of Process Configuration . . . 440
179 Graceful Degradation of Navigation Function . . . 443
180 DP/M Dedicated FBW Flight Control Configuration (Triple-Redundant Example) . . . 448
181 Transient Surge AC Voltage Step Function Loci Limits for MIL-STD-704A Category B Equipment . . . 452
182 Transient Surge DC Voltage Step Function Loci Limits for MIL-STD-704A Category B Equipment . . . 453
183 "Ultra-Reliable" PE Power Supply Functional Block Diagram . . . 457
184 Co-Located PE(s) and Associated Aircraft Device With Common Power Supply . . . 458
185 Physical PE Power Supply Allocation Alternatives . . . 459

## LIST OF TABLES

*Table* *Title* *Page*

1 DP/M PE Complexity Estimate . . . 19
2 Information Transfer Bus Traffic Decomposition . . . 36
3 Data Distribution Traffic (Bandwidth) Requirements . . . 36
4 BIU Commands (Instructions) . . . 73
5 BIU Interrupts . . . 73
6 Bus Interface Unit Functional Hardware Complexity Estimates . . . 103
7 Standard Format Instruction Set . . . 116
8 Extended Short Instruction Set . . . 120
9 DP/M Instruction Usage Analysis Summary . . . 122
10 Double Precision Operation Usage . . . 124
11 Complexity Factors . . . 149
12 Processor Complexity Summary . . . 152
13 Memory Use Versus Type Table . . . 154
14 PE Interrupt Vector Space Assignments . . . 156
15 Processing Element Internal Bus (I-BUS) Signals . . . 157
16 IOIU Commands (Instructions) . . . 192
17 IOIU Interrupts . . . 192
18 Input/Output Interface Unit Functional Hardware Complexity Estimates . . . 192
19 DP/M Memory Technology . . . 196
20 Current Memory Development . . . 198
21 DP/M Processor Technology . . . 201
22 Processing Element Complexity . . . 206
23 Projected 1980 DP/M PE Costs . . . 233
24 DP/M System Processing Tasks . . . 244
25 Loran Function Processing Tasks . . . 248

26 Loran SNS Input Data Example . . . . . 251
27 SNS Input Task Data Format . . . . . 256
28 Processor Instruction Simulation Algorithms . . . . . 271
29 BIU Simulation Input Data Specification Format . . . . . 282
30 CAW Operations . . . . . 287
31 LEX Preamble Entry Description . . . . . 309
32 GEX Time-Ordered Linked List Entry Description . . . . . 329
33 Assembly Language Model Development . . . . . 344
34 Close Air Support Mission/Attack Segment NARBS Mode Program Summary . . . . . 424
35 Generic Processing Operations and Instruction Count Summary . . . . . 428
36 Program Allocation Summary . . . . . 435
37 Glossary of Application Program Names . . . . . 436
38 Task ID, Memory, and Execution Time Summary . . . . . 437
39 Subfunction Information Summary . . . . . 438
40 DP/M System Failures and Detection Mechanisms . . . . . 446

# SECTION I

# INTRODUCTION AND SUMMARY

The purpose of the Distributed Processor Memory Architectures (DP/M) program was " . . to extend the DP M system concept work to a detailed system software and hardware design. The emphasis of the program is to develop and use a functional simulator that will aid and advance the design effort . . . " This report contains the results of this functional design effort and a description of the simulation and analysis tools that were developed. Pertinent design tradeoffs and the functional operation of the DP M hardware and software components are presented.

## A. DP/M SYSTEM CONCEPT AND OVERVIEW

The DP M system concept is essentially the use of a varying number of simple homogeneous processor memory elements (PEs) applicable to a wide range of avionic system processing problems. Architecturally, these PEs can be used as standalone uniprocessors or they can be configured in a distributed network as shown in Figure 1. Serial time-division-multiplex (TDM) buses interconnect the network. Two levels of busing are provided: a Global bus can interconnect each PE in a system network and a Local bus can interconnect multiple PEs clustered together to perform a given function. This cluster of PEs is referred to as an Affinity Group (AG). Input output (I O) for a given PE to an external device is via its local I O interface unit.

Four functional modules make up the DP M PE as shown in Figure 1. The Bus Interface Unit (BIU) is the basic TDM data transfer interface to the processor and memory. The BIU translates bit serial data into parallel data words and transfers data and status information to the PE processor and memory. The processor module is the instruction-sequencing and data-processing portion of the PE. Its computational capabilities are equivalent to present commercial and militarized minicomputers. The memory module provides necessary program instruction and data storage, and can be accessed by the other PE modules: the BIU, the processor, and the input output interface. Memory access is local to a PE, i.e., access by another PE or shared memory is not part of the DP M concept. The Input Output Interface Unit (IOIU) of the PE permits digital data, command, and status information transfers between the PE and the external devices in the avionics system.

The DP M concept is based upon the projected availability of low-cost, medium-performance PEs which can be used with TDM communication methods to provide a means of cost effective incremental system processing growth. This potential is enhanced by the use of common modules to achieve increased capability. The distribution of small, identical computing resources also promotes reliability by lessening system dependency on the correct operation of any single element. This natural distribution of processing tasks along avionic-function or sensor-partitioning lines allows a manageable system computational load and growth. The advantage of DP M in avionic system design is possible with the continued evolution of improvements offered by semiconductor technology. With the requirement for reduced size, weight, and power for airborne equipment, DP M offers a potential solution to these primary goals with its flexible building-block approach to avionic processing.

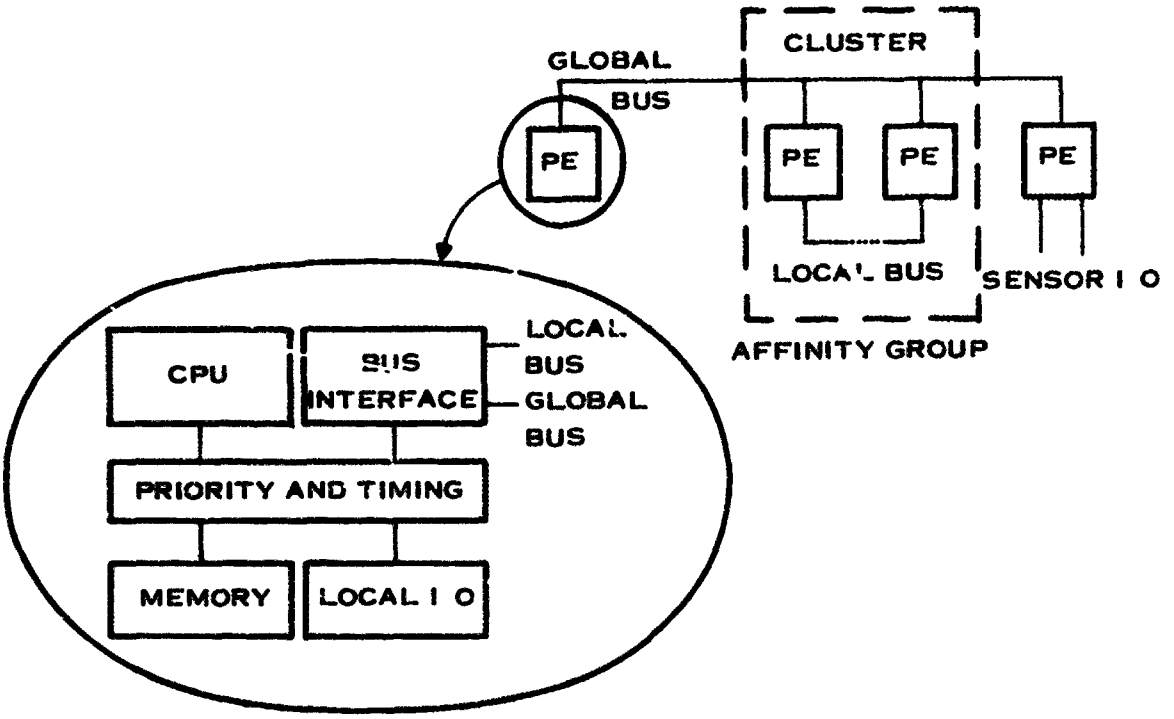



**Figure 1. DP/M System Architecture**

Specific tasks addressed in this program were:

- The functional design of the DP/M hardware and the development of a set of simulation and analysis tools
- The function design of the DP/M executive software
- The study of process construction methods for DP/M system software and hardware designs
- The examination of the role of and methods that could be used with DP/M to promote avionic system fault tolerance.

## B. SUMMARY OF DESIGN RESULTS

This subsection summarizes the following DP/M design activities and results:

DP M system network design

DP/M functional simulation system

PE processor, memory, and internal bus structure

PE Local and Global Bus Interface Units

PE I O Interface Unit

PE technology and cost projection

Executive software

Process-construction study

Fault-tolerance analyses.

### 1. DP M System Network Design

The inherent design flexibility of using small PEs to build different processing networks is a key feature of the DP M concept. One result of the system design effort was the realization that, while the concept is basically open-ended, the functional design of hardware and simulation programs had to initially assume a well-defined nature. It was not the intent that the small fixed functional resources of DP M solve every avionic processing problem. The homogeneous nature of DP M also stressed the careful review of all system design features, since all PEs were identical in capability. The aim of the system design was to allow as many options as to the use and application of the DP M as appeared feasible within the limits of future technology.

The baseline DP M design has the following features:

A dual redundant Global bus system for inter-PE communication

A single Local bus per Affinity Group of PEs

Functionally federated PEs or groups of PEs devoted to the processing of a given avionic function.

Both the Local and Global bus used a distributed round-robin bus control scheme where each PE interconnected to a bus has a predefined allocation of bus usage. This allocation mechanism is programmable and, as such, super commutation of bus messages can be implemented. A logical bus message identification header is provided in which a message can be broadcast to multiple recipients. These bus protocol features are provided for both the Global and Local buses.

The Local bus facility is primarily meant to allow the creation of AGs from PEs. Because of the likely closeness of PEs within an AG, a single Local bus interface per PE is provided. The Global bus interface provides for a dual redundant TDM communication line for each PE. Every PE on the Global bus can participate in primary bus transfers and simultaneously listen on the secondary bus for a "switch to back-up bus" command.

### 2. DP/M Functional Simulation System

The development and delivery of a set of simulation computer programs that model both a distributed network of PEs and the key design and performance features of the DP M PE was a

key result of the design effort. Both simulators are capable of modeling discrete events using time-order lists. The System Network Simulator (SNS) synthetically models the asynchronous operation of multiple PEs that are executing programs and a distributed round-robin bus control on the Local and Global PE bus interfaces. Hardware models allow simulation of the programmable bus control over a user-specified number of interconnected PEs. Models of executive software routines have been included to permit the scheduling of programs defined as directed graphs composed of processing tasks. An extensive set of reports was defined to analyze the results of network simulation runs. These reports provide bus loading per user-defined time interval and individual PE processor execution-time loading and memory usage. Similar resource usage reports are available for different bus, PE, and avionic software activities. The capabilities of the SNS are also a vital part of the process-construction methodology for developing real-time software for DP/M.

In contrast to the high-level traffic simulation available with the SNS, the PE simulator allows for detailed emulation of the four DP/M PE sub-elements. The processor simulation model executes each PE instruction and alters all programmer-visible registers. The BIU hardware model allows for simulation at the register transfer level, and the IOIU model provides for a high-level functional simulation of data-channel transfers, interval timers, and the interrupt structure. Facilities are also provided to define and generate interrupt stimuli to the PE, and multiple requests for access to PE memory are accurately simulated by priority-resolving logic and memory-cycle stealing. Debug features are included to permit detailed tracing of instruction and BIU operations. Memory line printer dumps, execution flow path counters, and static and dynamic histograms are available for post-simulation analysis of PE operations. Interfaces are also defined to permit definition of Local and Global bus message data transfer sequences and error conditions, as well as the use of detailed simulation models of external input/output devices.

## 3. PE Processor, Memory, and Internal Bus Structure

The PE processor design specified a 16-bit, 8-register file microprocessor with microprogram control. Careful determination of required data/control signals indicated that the processor can be implemented in either one or perhaps two 48-pin LSI (large scale integration) packages (a data processor chip and a control chip). Other features of the processor include a uniform set of instruction fields that simplify instruction decoding and reduce device complexity. Functionally, program memory is addressable to 65K words; however, most applications are expected to require 4 to 8K words of program/data storage. A set of forty 16- and 32-bit instructions have been specified, and primary emphasis has been placed on maximizing the efficiency of those high-frequency operations typically found in avionic software. An extensive set of 16-bit instructions is provided to conserve memory, an important factor, since memory will eventually dominate the cost of the PE. A standardized set of 80 internal PE bus (I-BUS) signals (including 20 for power and ground) was defined that facilitate intra-PE data transfer and control operations. An added benefit of the I-BUS is the ability to combine and/or intermix PE modules implemented in different technologies. The proposed DP/M design is easily implemented with current semiconductor devices and, as future LSI devices become available, the PE can use such devices so long as the standard I-BUS signals and interface conventions are observed. A simplified programmer maintenance panel interface is provided via the I-BUS, and the I-BUS also offers a method of providing system fault isolation and built-in test (BIT) for the PE.

## 4. PE Local and Global Bus Interface Unit

An approach that emphasized functional modularity was used in the design specification of the PE BIU hardware. This was particularly important since each PE in a DPM network is required to have two levels of bus interface: one global and one local. The BIU devices support a distributed round-robin bus protocol scheme with message broadcast capability to multiple PEs. The BIU contains the necessary hardware to associatively match logical input message identification headers and automatically vector these messages into software-selectable PE memory buffers. This data-vectoring feature relieves the processor from continually setting up and servicing the BIU hardware, an important response time feature when both the Local and Global buses are processing messages. The primary BIU hardware devices for the Local and Global buses are a common design with signals set aside on the device to serve as function-selection pins, i.e., these signals dictate the operations required to make the device function as either the Global or Local BIU, and, consequently, increase the use of the same common devices.

Multiple-device partitioning was recommended for the BIU hardware functions. Those devices that were identified included:

- A bus interface logic unit (BILU) that is responsible for interfacing the PE to a DPM bus
- A bus interface translation unit (BITU) which converts serial bi-phase to parallel NRZ data
- A redundant bus management unit (RBMU) which is used to implement the RBMU function.

## 5. PE I/O Interface Unit

The PE IOIU provides the basic PE external data transfer facilities, interrupt-masking and priority-resolving logic, interval timer, and internal PE clock. Two LSI alternatives were identified for these IOIU functions. One approach was to use a single, complex, large-scale integration device to implement most of the IOIU functions. An alternate approach was to establish a modular family of devices for the IOIU functions. This family included the following part types:

- A two-device autonomous data transfer channel
- An input/output data interface device to provide data path buffering between the PE I-BUS and I/O data lines
- An I-BUS master interface device which performs I-BUS control functions
- An I-BUS slave interface device used for passive device (e.g., memory) and I-BUS interface
- An interrupt interface device which provides interrupt marking, priority, and control functions
- A programmable interval timer.

## 6. PE Technology and Cost Projection

The investigation of candidate implementation technologies and the derivation of a cost projection for the DPM considered several items. The examination of LSI technologies revealed the following information:

Several high-device-density, low-power LSI technologies offer the necessary performance and size characteristics for DP/M applications. Prime candidates include NMOS, $I^2L$, and CMOS.

The wide operating temperature range for military applications must be considered in the selection of device technology. Avionic equipment using the DP/M must operate in temperature environments that can vary from −55°C to +100°C; this implies that devices inside those equipments will experience temperatures from −55° to +125°C.

Technology producibility affects device yield and, eventually, end-item cost. Ideally, the technology selected for the DP/M should be used for high-volume commercial production. This would allow significant cost advantages and would minimize the problem of obtaining multiple sources.

NMOS possesses the best performance density characteristics of the presently mature LSI technologies; however, its ability to operate above +90°C is unclear. Two high-density, low-power LSI technologies capable of wide operating temperature ranges are Integrated Injection Logic ($I^2L$) and Complementary MOS (CMOS). Both technologies offer PE implementation and are capable of operating over a wide temperature range. Additional benefits include a natural tolerance of power-supply voltage variations and potential candidacy for nuclear radiation hardening. While $I^2L$ is a newer technology, its bipolar technology characteristics have the potential for paralleling the TTL 5400/7400 device family where the same manufacturing processes can be used to make both military as well as commercial temperature range parts. The manufacturing process for $I^2L$ wafers appears to have fewer steps than CMOS, and $I^2L$ offers a 40-percent device-density improvement over CMOS.

A partitioning of functions and an estimate of the device complexity of the DP/M PE was derived and the results are summarized in Table I. These data were the basis for a cost projection for the PE in the 1980 time period. A commercial grade (0°C to +70°C) DP/M PE was estimated to cost approximately $350 in quantities of 5,000, while the military grade equivalent was estimated to cost $1,745. Both estimates offer significant improvement over the current costs associated with avionic processors. The large variant in cost for a military-grade component is associated with the additional operating temperature range, hermetic package, inspection, burn-in, testing, and reliability requirements imposed by military specifications on components.

## 7. Executive Software

The executive software design effort resulted in the detailed functional specification of a two level control structure. A set of common routines was defined for the Global Executive (GEX) and Local Executive (LEX) functions. The GEX is responsible for system scheduling and mode control and the Local Executive (LEX) provides the necessary control function for tasks assigned to a given PE. The organization of both executives is based upon a table-driven control structure philosophy to permit their use and adaptation in different PE network configurations. Detailed simulation models of major GEX and LEX routines were developed and incorporated into the SNS.

**TABLE I. DP/M PE COMPLEXITY ESTIMATE***

| Functional Unit | Complexity (gates/bits) (X1000) | Packages | | Pins/Package | Power (watts) |
|---|---|---|---|---|---|
| | | Type | Total | | |
| Processor | 3 to 4 | 1 to 2 | 1 to 2 | 48 | 0.7 |
| Memory controller | 0.6 to 0.8 | 1 | 1 | 48 | 0.2 |
| Memory | 128 to 132 | 1 | 8 to 9 | 20 | 3.2 |
| Bus interface | 4 to 5 | 3 | 8 | 14, 20, 48 | 1.7 |
| I/O interface | 1 to 1.5 | 2 to 6 | 6 to 9 | 16, 18, 24, 28, 40, 48 | 1.9 to 2.1 |

Note. See Section VI for additional background data.

## 8. Process Construction Study

The process construction study addressed the requirements and defined a methodology to be used in designing real-time software for DP M. The structure and characteristics of software for a distributed network of PEs were examined and representations of these processes were identified. These structures include:

the data flow graph

the control flow graph

the program directed graph

the maximally-parallel predecessor successor graph.

A procedure was defined for real-time system software design, development, and validation that uses successive levels of model development and four levels of simulation, including:

system network simulation

functional simulation

instruction level simulation

analytic/hybrid simulation.

Three levels of process construction were identified, the Basic Process Constructor, the Intermediate Process Constructor, and the Production Process Constructor. The emphasis of the DP M design effort was on the specification of requirements for the Basic Process Constructor. Nine basic steps were identified in the Basic Process Constructor, and appropriate algorithms were identified for each step. Certain procedures were amenable to automated techniques and, as time permitted, analysis programs were written for the following functions:

- A topological sort that analyzes the input output precedence relations between tasks of a program model and generates identification numbers for all program tasks.
- A path matrix constructor that analyzes the structure of program task graphs and forms a matrix of task interrelationship
- An execution path generator which constructs all alternative execution paths throughout a program.

Data produced by the preceding programs were used to generate DPM PE execution time and memory storage requirements for avionic mission software models and to verify the proper organization of the software models. Algorithms for allocation of avionic software to the PE network, based on integer programming, cluster analysis, and a heuristic method were identified and their characteristics discussed.

A functional specification of the recommended DPM process construction procedure was defined and included:

Process construction algorithms

Input/output data sets

Hardware and software implementation requirements

To gain an insight into the requirements of DPM process construction, a limited case study was performed. Computerized methods were used to estimate mission software execution time and memory requirements on the DPM PE and to manipulate a data base of bus and I/O signals. A summary of this case study is included in Subsection IX.D and Appendix B.

### 9. Fault Tolerance Analysis Activities

The role of DPM with respect to fault tolerance in an avionic system was examined. Several key observations were made. The expected reliability of LSI DPM devices when compared to the typical failure rates of avionic sensors is quite high and, consequently, DPM is not expected to be a source of faults when deployed in avionic systems. The DPM is amenable to physical and functional avionic fault-tolerance techniques. Physical fault tolerance is achievable via redundancy techniques such as those used in fly-by-wire flight-control systems. Functional fault tolerance includes software techniques for achieving degraded modes of operation once an error has been detected. Features have been included in the PE processor, BIU, and memory to aid in error detection and recovery activities. The power supply requirements and transient conditions are known in an avionics environment, and the use of an ultra-reliable power source for DPM has been investigated.

## C. REPORT OVERVIEW

Subsequent sections of this report discuss the following topics:

System design considerations

Information transfer bussing system design

Processor memory design

External I/O device interface

PE implementation considerations

Functional simulation system

Executive design and system operation

Process construction study

Fault tolerance analysis.

## SECTION II

## SYSTEM DESIGN CONSIDERATIONS

This section discusses the basic considerations used to derive the functional design of the DP/M system modules. As such, it is intended to provide a high-level overview of those thoughts that influenced the design process throughout the DP/M program. A review of the system configurations for DP/M is presented, and an overview of the DP/M system communication approach and a review of design alternatives conclude the section.

### A. DESIGN CONCEPTS

The task addressed during the DP/M program was to specify a functional detailed design for the DP/M hardware and produce a set of flexible software simulation/analysis tools. This design was to be based upon those avionic requirements identified in previous Air Force studies. For the DP/M design effort, a set of concepts was identified and served as guidelines throughout the design process. These guidelines were by necessity general in nature, but their purpose was to ensure that the output of the DP/M design was as applicable as possible to projected Air Force requirements.

#### 1. All-Semiconductor Design Concept

One of the original DP/M concepts was the eventual realization of an all-semiconductor design, and distributed dedicated processor/memory pairs lends itself to an all-semiconductor implementation. A major advantage of this concept is the use of LSI technology for the 1978–80 time period for a DP/M system. Projections as to the practicality and cost effectiveness of LSI vary; however, trends in the early and middle 1970's indicate LSI manufacturing techniques are maturing and that LSI semiconductor devices are being economically produced. Historically, memory devices have been the leader in LSI device production due to their large commercial market demand. An encouraging feature of the 1973 and 1974 semiconductor product lines was a variety of LSI logic functions such as 4- and 8-bit microprocessor chip sets. The commercial market availability of such memory and logic devices gives a continued indication that an all-LSI DP/M for the 1978–80 time frame is indeed feasible.

#### 2. Architecture Selection Concept

The basic organization of the DP/M processing element (PE) has been established in previous AFAL sponsored work and was shown in Figure 1. Several thoughts guided the actual design specification of the PE processor, memory, and Bus Interface Unit modules.

The processor module will quite likely be a one- or two-chip LSI central processing unit (CPU) device in the DP/M time frame. Present semiconductor LSI chip sets offer a variety of 4-, 8-, and 12-bit microprocessor or microcomputer architectures while 16-bit microcomputers are projected for 1975. These choices in the data path bit width of the processor architecture are the result of current market demands and the present state of LSI technology manufacturing processes. Several previous Air Force studies have recommended that a large portion of the avionic processing problem space can be solved with a 16-bit computer architecture. This 16-bit data-processing organization also parallels the wide number of available commercial 16-bit

mini-computers and their planned "micro equivalents" and is the most likely and predominant organization of micro-class computers for the DP M time period. The goal of the DP M processor design was to produce a detailed specification and simulation program model that would be applicable to a number of avionics computational problems. Now that this design has been completed, AFAL has both a design and the detailed simulation tools that can serve as a departure point suitable for evaluating different processor architecture alternatives.

The semiconductor nature of the DP M had an impact on the PE and its memory design. The primary concern was the volatility of semiconductor memories (i.e., loss of data when power is removed). The three candidate solutions to this problem are: (1) the use of a non-volatile memory device such as MNOS, (2) the use of a reliable and continuous DP M power source, or (3) the use of non-volatile fixed memory device such as Read Only Memories (ROMs). The goal of the DP M functional design was to investigate candidate solutions to this problem and to ensure that the PE functional architecture did not restrict the use of different memory technologies.

The BIU module was the key component used to build a network of distributed PEs. The specification of its organization had to consider such items as expandability, efficiency, and fault detection. The dual-level global local nature of the DP M bus philosophy is the primary means of expanding the processing power of the DP M and building different network configurations. Each PE is capable of having this dual-level bus interface and, as such, the BIU represented one of the more common and potentially complex hardware modules. It was recognized that the servicing of input output message transfers by the processor via the BIU must be efficient (i.e., minimize processor throughput degradation). The BIU's role as the primary inter-PE communication path also dictated that the design allow for fault detection and recovery mechanisms within the DP/M network.

## 3. Compatibility Concept

The DP M program recognized that its system design must be cognizant of and compatible with Air Force standards that would be applicable in the 1978 - 1980 avionics time frame. While it is admittedly difficult to project the exact nature of future avionic systems and Air Force standards for components in these systems, such present standards as those for environmental and electrical power interface were considered in the DP M system design. It was the aim of the system design that evolving testing and maintainability philosophies for Air Force equipment should be encouraged by use of DP M system components. This emphasis on compatibility with Air Force standards also dictated that methods should be identified during the design to encourage compliance with these specifications.

Consider, for example, the accepted way in which parts are currently selected for use in electronic equipment. The continued evolution of microcircuit technology has resulted in the use of the Approved Parts List typified by MIL-STD-701 and stringent integrated circuit standards such as MIL-M-38510 or MIL-STD-883 to guide militarized equipment parts selection. The intent of these specifications is to offer a common set of approved parts manufactured and inspected to the Government's satisfaction and to logistically reduce a potential proliferation of nearly identical part types. In its early stages, this approach was well accepted by both the semiconductor manufacturers and militarized-equipment designers. However, trends such as the one shown in Figure 2 indicate that military demands on microcircuit technology are being diminished when compared to non-military users. As the capacity of semiconductor

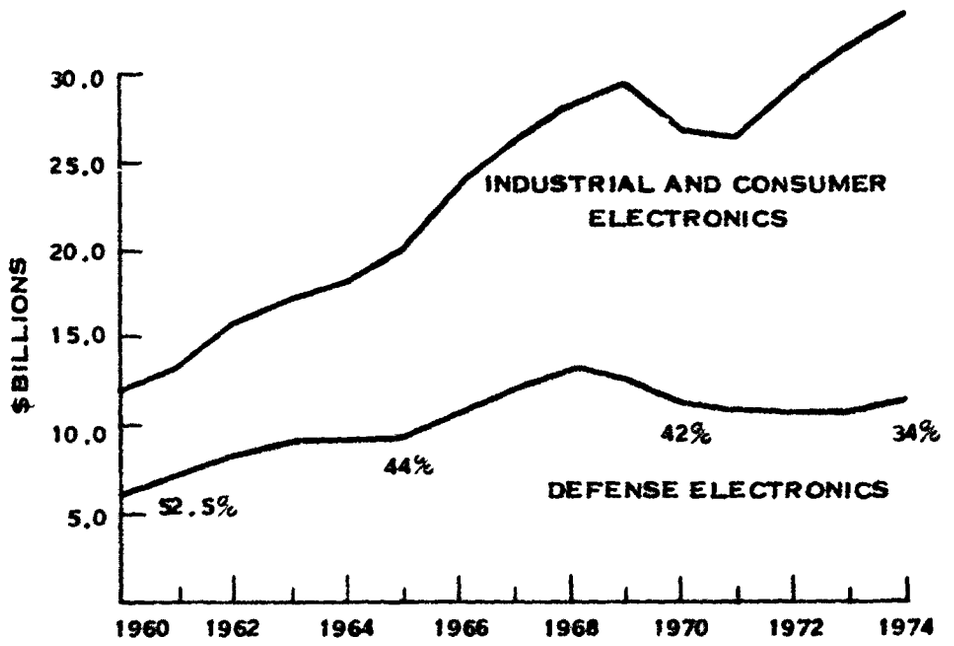



**Figure 2. U.S. Electronics Industry Trends**

manufacturers is absorbed by the larger dollar non-military markets, certain items on the Approved Parts List (APL) are discontinued if their production demand cannot keep pace with the sales of non-APL devices. Ideally, the most desirable solution would be parts that could satisfy both military and non-military specifications, thereby creating a sufficiently large demand for more production of such "dual-purpose" devices. Likewise, a military specification part that is usable in a sufficient number of different military applications can create a suitable demand for high-volume production. The DP/M concept has the potential to fulfill this multiple-application requirement with its low-cost LSI PE modules. It was the goal of the DP/M design effort to identify ways in which these compatibility concepts can be encouraged.

## 4. Usability Concept

A continued emphasis on usability dominated the design activity, as did the consideration given to eventual DP/M deployment in avionic systems. As the expected costs of the DP/M hardware decrease, the total life cycle costs associated with using the DP/M in system design, development, test and evaluation (DDT&E) as well as field deployment will be magnified by comparisor. An obvious non-military parallel exists today when one considers the labor cost of design engineering as compared to commercial minicomputers that can be purchased for $1,500 to $3,000. The use of the system (its design, check-out, and maintenance) clearly dominates the cost of the hardware. The concept of a usable DP/M system influenced each phase of the hardware and software design.

Applying this concept of usability influenced the functional simulator design effort as well as the process construction study and functional hardware definition. To be useful, the System Network Simulator structure had to be flexible and allow for a variable number of topologically interconnected PEs to interact over IDM buses. The usefulness of the network level hardware model was to be complemented by a more detailed PE simulator. The goal of the PE simulation was to allow detailed evaluation of the PE while executing programs, transferring serial bus data, and transferring I/O data. Furthermore, this functional simulation system had to be suitable for a variety of paper analytical design studies. This dual-level simulation capability was to be further used by being incorporated by the process construction activity in specifying a method for the DP/M Process Constructor.

Usability also influenced the PE hardware design. With respect to the DP/M hardware, it quickly translates to "how easily can I apply this hardware to different applications" and "how can I test and check-out my design." This emphasis on testability, fault isolation, and simplified maintenance concepts was to guide the design effort and to ensure a usable DP/M design was produced.

## 5. Modularity Concept

The concept of modularity was another key design consideration for DP/M. Modularity influenced the ease of system application and eventual system cost. Consider, for example, the DP/M PE. A specification of too high a level of integration of PE functions in one LSI device can potentially complicate the hardware implementation while at the same time reduce manufacturing yield and increase per-unit cost. A futher disadvantage is possible in that the application of the device in other systems can be often limited because of its special-purpose design. In contrast, a simple approach using low levels of integration can cause an explosion of part types that complicate system design and increase physical size and logistic problems. Unfortunately, opinions as to the ideal level of the modularity will differ with each application and system designer. For the DP/M design, the general concept was that functions should be implemented with modules which will allow for reasonable incremental growth with respect to capability.

One recognized method of encouraging modularity is the use of a standard interface. A standard interface can be used to allow incremental growth and expansion capability as well as to provide an orderly method of technology involvement for the different DP/M system modules. The interface signals and method remain constant and allow different technologies to be used, providing they meet these interface requirements. Standards like ARINC and FIA have long used such methods to provide growth and technological evolvement for their respective system components.

To a degree, modularity was recognized to affect the application and use of all avionic components. To illustrate, if the choice of processing growth alternatives was found to be in $100,000 increments and 40-pound boxes, this "coarse" level of modularity negates its wide application. However, if the cost and size of incremental growth and expansion were similar to that envisioned for LSI DP/M devices, the avionics architect has a large degree of flexibility in applying more computerized resources to his problem. This wide use of the same common modules has other important benefits, all centered around the idea of replication. Electronic equipment and semiconductor manufacturers such as Texas Instruments use this high level of

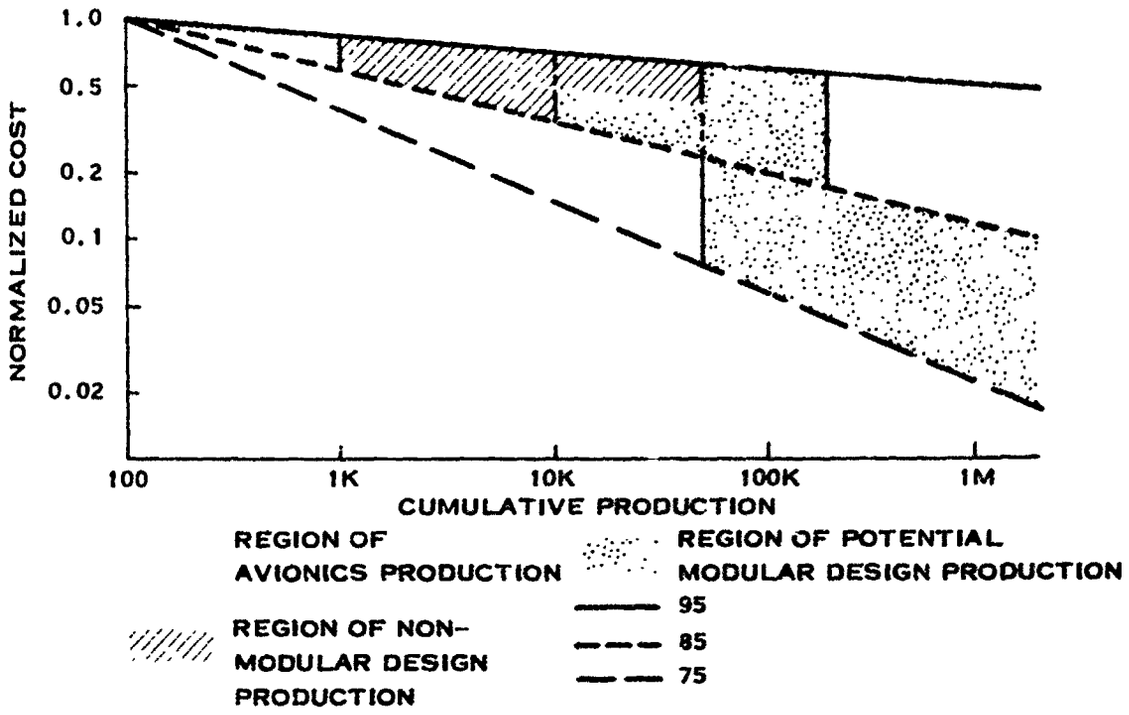



**Figure 3. Modularity Learning Curve Effects**

replication of a common component (or process) to lower the acquisition cost of a module to a system user. The benefits of maximizing the modularity of an equipment can be shown by a quick look at the effects of learning curves on modularity and replication on cost.

Figure 3 shows a family of learning curves and the regions on these curves where production avionic and weapon systems may fall. A learning curve is based on the historical fact that when production increases, cost decreases. A 95-percent curve implies that when production doubles, the unit cost decreases by 5 percent. The figure shows 95-, 85-, and 75-percent curves. Historically, a slope of 95 percent is achievable by worker training, tooling, and manufacturing flow improvement. A slope of 85 percent is achievable through automation, and 75 percent is achievable only through basic process improvement, product evolution and yield improvement. Production of most military equipment typically falls in the 95- to 85-percent region and frequently does not even achieve that level because of insufficient firm volume levels to justify automation and because of broken production lots, due to funding phases, which necessitate employee reassignment.

Examining the curve shows the potential value of modularization with DP/M technology. A typical nonmodular design might have a total production varying from a few thousand to as many as 50,000 units. If several systems use the same modules, total production could approach several hundred thousand units. Further, the broader application base lessens the risk of automation investment and allows sooner entry into the 85-percent total inventory requirements, and AGE and training costs are reduced.

The greatest reduction in cost will be achieved only if the slope of the learning curve can go lower than 85 percent. The question is, can it and, if so, how? The cost curves for digital integrated circuits, over the past several years, have displayed 75 percent or better slope. The reason is essentially that of "modularity." Almost all digital equipment uses a few basic families of devices. This standardization has resulted in the volume base to achieve cost improvement. Notice that standardization has not implied stagnation. The number of devices in a given line increases almost daily and different packaging and screening processes yield devices for both severe and nonsevere environments. All devices, however, use the same processes and the "shared experience" results in lower unit cost for each member of the family.

It is this same goal of maximum modularity and applicability that was considered during the DP/M hardware and software design.

## B. SYSTEM APPLICATIONS

The projected size, weight, and minicomputer-like computation capabilities of the DP/M modules permit their use in several avionic applications. Candidate applications were identified in which DP/M modules could be used, and each role was examined for the requirements it would impose on the DP/M design. Consideration was given to the types of computation (arithmetic, logical, data manipulation), algorithm execution time constraints, and special deployment requirements such as maintenance and testing features. This requirement search was by necessity brief, but it identified potential applications and emphasized a set of requirements for the DP/M design. The three roles considered for the DP/M were:

The stand-alone computation unit

The smart sensor or standard interface unit

The distributed function configuration.

### 1. Stand-Alone Computation Unit

The stand-alone DP/M configuration is a classical uniprocessor architecture where the PE processor, memory, and I/O modules are deployed as a limited-capability microcomputer. Current avionic equipment using such microcomputers includes radio and TACAN navigation sets, digital air data computers, and aircraft ground proximity warning systems. Algorithms for these functions such as the air data equations are straight-forward, and until recently have been primarily implemented using either mechanical or analog techniques. The projected cost and size of DP/M-like modules open up several new applications including missile digital autopilots, control and display logic controllers, and built-in test (BIT) and diagnostic sequencers. This use of a few LSI devices to add self-diagnosis, test, and maintenance capabilities offers a significant potential in reducing future maintenance costs of avionic equipments.

### 2. Smart Sensor or Standard Interface Unit

A natural follow-on to the stand-alone configuration is the use of DP/M module to pre-process data within a sensor and provide a standard interface to other avionic equipments. In the preprocessor role, data is converted to a common format for use by other avionic system elements. A review of some examples found in previous avionic study efforts will highlight the requirements within this application, and these requirements typify the operations normally found in avionic computation.

The present DAIS (Digital Avionics Information System) concept is taking one step of providing a standard interface by converting all avionic subsystem signals into a digital format within a remote terminal, and transmitting these converted signals over a multiplexed data bus to the DAIS processors. In its present form, the remote terminal converts and, in some instances, packs this signal data, but does not attempt to preprocess the data for digestion by the DAIS computers. An examination of the DAIS sensor signal list shows data types that include discretes (to be packed/unpacked), binary 1's and 2's complement data, weighted or scaled binary (e.g., 1 bit equals 1.5 degrees), and binary coded decimal (BCD). The core processors accept this data from the remote terminal and perform the packing/unpacking and conversion to the proper format (predominantly floating point) for use by the different avionics programs. This requires executive software activity to schedule this task as well as computational resources to perform the conversion. The use of an inexpensive DP/M PE to assist the relatively more complex and expensive DAIS computing resource offers a cost-effective method of system growth and expansion.

Examples of DAIS-like avionic subsystems that require preprocessing include the Loran receiver and the weapon station. The examined Loran receiver outputs two 26-bit time-difference quantities represented as six 4-bit BCD characters plus two status bits. These data must be decoded and converted into a format suitable for use by the Loran navigation algorithms. Likewise, the receiver expects from the computer binary velocity aiding information made up of 11 data bits, sign bit, and two status bits, all in one's complement form, while other outputs are BCD encoded. While the information formatting is not complicated, it is well suited to preprocessing by DP/M-like microprocessors. Another example of a subsystem using BCD data inputs is the DAIS hot bench weapon station in which weapon count and station identification are to be transferred in this coding scheme.

One example of a demanding preprocessing function is associated with extracting data from synchros and resolvers. Usually, these devices output two numbers that represent the magnitude of the sine and cosine of an angle. To determine the actual angle for use by different avionic subfunctions, the quotient of the sine and cosine can be formed to determine the tangent, and then the arctangent of this number is taken to determine the angle. This processing is obviously more mathematically complex than the first example but still well within the capabilities of a DP/M PE.

A similar potential application was identified by examining an existing airborne emitter location system that interfaced with an inertial navigation subsystem. The heart of the system used a high-speed digital processor for hostile threat classification, prioritization, and location calculations. The inertial navigation subsystem was used to supply aircraft roll, pitch, and yaw data as well as velocity north and velocity east information. This information was sampled by the digital processor at data rates of 5 and 25 Hz, and its processing was a high-priority task to be performed. Each data item was input as a 16-bit word in the format shown in Figure 4.

**Figure 4. Example Data Input**

Processing of the data included extracting the hardware status bits, scaling the remaining sine and cosine, computing the square root of the sum of the squares of the sine and cosine, checking the result for a value within proper limits and normalizing the result if necessary. Other operations included validity checking of computed results. While the computation associated with these operations again is not difficult, there is a potential sacrifice of processing devoted to threat classification and location calculations when overhead is devoted to input preprocessing. In a dense environment of hostile emitters, this could result in less than maximum capability system performance. Once again, DP/M-like microcomputers could offer an economical alternative to multiprogramming the larger and faster main processor by having current data available in the proper format and at the proper time.

It follows that the next generation of sensors will most likely include some type of programmable logic unit that can provide data in a standard format suitable for use by a given application system. The DP/M PE design offers this capability to the avionic system architect.

Benefits obtainable with this smart sensor approach include a standardized and flexible peripheral interface that has the potential to improve reliability while minimizing the impact on system cost. The primary advantage is the establishment of a simplified, programmable data communication interface that permits orderly growth and a natural distribution of processing within the system. In addition, this type of dedicated low-cost processing offers the potential to simplify check-out and maintenance of avionic functions by adding computation intelligence in the sensor.

This use of the DP/M PE also has another benefit in that the addition of a set of LSI chips to the peripheral components can significantly increase the usability of the device while not degrading its reliability. Reliability is a key factor in avionic system design since a reduction in reliability implies more failures can occur. Each failure requires a maintenance action which in turn increases the overall system life cycle cost to the Air Force.

Projection of the MTBF of DP/M-like LSI chips can be made to emphasize this point. A previous Air Force study made a rough prediction based on rules typified in RADC-TR-69-350, "Failure Rate Prediction for Complex Bipolar Microcircuits." Using extrapolation techniques similar to those described in this report, the failure rates of electrically stressed militarized LSI chips with as many as 4,000 gates was estimated to be on the order of magnitude of 0.02 percent per 1000 hours. This ratio implies each device would have a calculated MTBF of five million hours. When compared to the predicted MTBFs of sensors such as gyros and air data sensors that range roughly from one to 10,000 hours, it is obvious that the addition of a few high-density LSI devices to the sensor peripheral will have a negligible impact on failure rates.

### 3. Distributed Function Configuration

The third application of the DP/M components is in an interconnected network of PEs via IDM buses that operate as a distributed function processing system. A block diagram of a general DP/M system configuration is shown in Figure 5. This figure is not intended to depict any particular DP/M network but instead shows interconnection of major components.

Aircraft sensors and actuators communicate with a given DP/M PE via a digital interface unit. Input/output between the PE and external device is in a digital format, with external signal conversion for the sensor/actuator at the digital interface. For the general case, the DP/M system functions as a federated and "dedicated at the sensor" processing system. Those PEs and Affinity

Groups that are not required to interface to the aircraft sensor/actuator system function independently and communicate over the Global bus system. The distributed nature of DP/M also permits geographic separation of PEs and Affinity Groups throughout the airframe.

## C. OVERVIEW OF SYSTEM COMMUNICATION/INTERCONNECTION FACILITIES DESIGN

The DP/M system interconnection and communication facility design reflects a multi-level/multi-functional data-transfer philosophy which supports digital data communications at two basically distinct interface levels: (1) inter-PE (i.e., intra-DP/M system), and (2) PE-to-external device (i.e., between the DP/M system and external aircraft subsystem equipments). This chosen approach segregates inter-PE or inter-program information exchange from "input/output" (I/O) data transfers between PEs and external devices, as shown in Figure 6. The two interface facilities are different in function and operation and represent a "best-fit" approach to functionally independent interface requirements while also maximizing system hardware efficiency and simplicity. This approach was formulated from rationale discussed in the following paragraphs.

The primary alternative addressed in the DP/M communications structure design dealt with the method of communicating between PEs (programs) and external I/O devices. In essence, the basic choices were to either make each I/O device directly accessible to all or a subset of PEs (e.g., via a common busing arrangement) or allow direct communication between PEs and I/O devices on an exclusive one-to-one basis. The latter (chosen) approach has one apparent

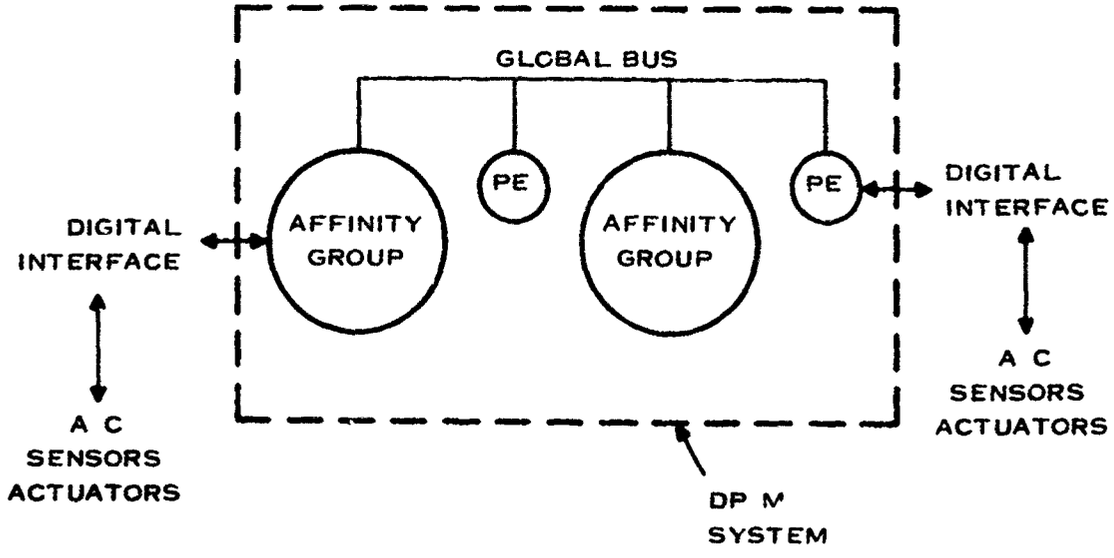



**Figure 5. General DP/M System Network Configuration**

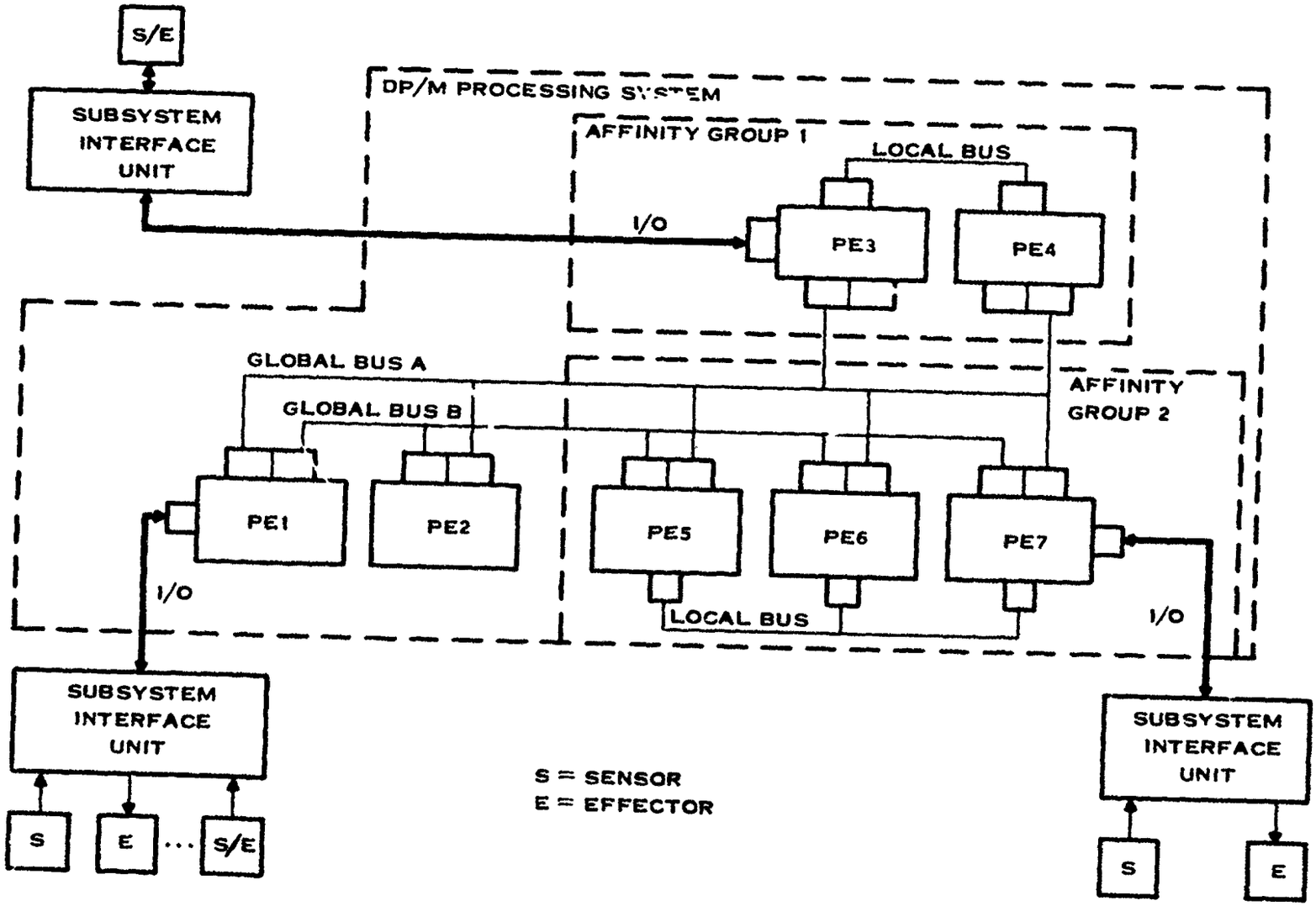



Figure 6. System Interconnection and Communication Structure (Example)

disadvantage with respect to the former approach, the disadvantage is due to the effective dedicating of devices to single PEs which may perhaps affect both system-level functionality and reliability. However, this apparent disadvantage is somewhat, if not completely, relieved because the device can become accessible to all members of the DPPM system (including other external devices as well as PEs) indirectly via the PE to which it is physically attached. Each PE is connected to all other system PEs via the Global bus. The PE, then, becomes a "Bus Interface Unit" by which all I/O data to from a particular external device is routed within the integrated avionics information system.

This I/O approach not only offers functional equivalence with the alternate approach, but also reduces system hardware complexity. External device interfacing is greatly simplified in that PE-to-device communications can be performed via a simple, conventional word-parallel digital interface amenable to rigid standardization. This simple interfacing technique is allowed due to the localization of the external device and its "interface PE." Since the PE may be assigned system functional processing tasks in addition to those of a mere "I/O device handler/preprocessor," the bus interface hardware used by the PE in its system processing role can also be used to distribute I/O device data throughout the system, if required by other processing functions (PEs). Thus, a common hardware resource is used, or "shared," by both processing and external (I/O) device entities.

In addition to the above advantages, the chosen approach also affords a greater degree of control and performance in the distributed function processing environment since the access to and, hence, operation of each external device is controlled by an "intelligent" member of the processing system. The same consideration of overlapped use of resources is applicable here as previously mentioned, only with respect to usage of the data-processing portion of the PE. Again, the external device interface hardware burden is lessened and, moreover, the integration of "smart sensors" into the information system is readily accommodated. In summary, the multi-level communication structure design offers the potentially lowest-cost approach that is compatible with DPPM system operational objectives without impairing system functionality and reliability.

## SECTION III

## INFORMATION TRANSFER BUSES

### A. OVERVIEW

All DPM intra-system communications are accomplished via a simple set of "party-line" data buses, the number of which depends upon system requirements of functionality and reliability. This inter-PE busing facility is structured to facilitate transmission of common, interprocess ("global") data elements and system control information (e.g., aircraft state vector and system executive control information) to all PEs. Also, the busing structure supports the transmission of uniquely "localized" data items between PEs associated with the performance of a single function or process, or between closely related processes, referred to hereafter as "affinity groups." The information transfer facility is, then, a hybrid structure consisting of a global communications bus and a variable number of dedicated "local" communications bus links. The number of individual local links depends upon individual system-processing requirements. These buses are incorporated in an inter-PE communication facility structural design which resulted from the following inter-PE communication system design-intent philosophy:

A global "party-line" facility must be available to all PEs for transfer of system control and interfunctional process data. A global "broadcast" capability is desirable to accommodate multiple-destination transmissions with minimum bus bandwidth usage.

A separate "local" communication facility for intrafunctional data transfers is desirable to:

Increase system-expansion capability by alleviating possibility of global bus traffic overflow

Increase throughput expandability of the total system by localizing high-data-rate transfers associated with the performance of an individual process. These transfers can be associated with intra-functional process data-passing or distribution of external device (I/O) data among a group of common I/O data subscribers (I/O data is typically related to a unique, specific avionic function).

Offer improved system response time associated with time-critical data transfers for unique processing requirements by transmitting critical data via high-priority, low-traffic localized bus.

Offer backup capability for global bus connection.

Since processing power can be scarce in a microprocessor-implemented system, bus-message-transfer activity should not significantly affect the operation of PEs not involved in the exchange and should require minimum program intervention control in affected PEs, i.e., the busing facility element(s) should augment the processor (software) in the handling of inter-PE data transfers.

Busing facilities should allow maximum data transfer efficiency for varying system communication requirements.

Preceding page blank

Buses should support physical distribution; i.e., the busing facility should permit and facilitate inter-PE communications over diverse path lengths (up to 300 feet)

Busing system design should allow low-risk technology/implementation within LSI constraints.

The Busing system should facilitate modular system growth, or reconfiguration, while incorporating a set of homogeneous communications elements (devices) of as few types as possible.

## B. SYSTEM REQUIREMENTS AND CONSIDERATIONS

The DP/M communications facility structural design was developed around the generalized design goals discussed previously. Details of the communications facility elements design(s) were driven by the following unique requirements generated from a top-down analysis of a representative DP/M avionic system operational environment.

### 1. Distributed Systems Environment

One of the pristine system-level criteria driving the DP/M Processing Element conceptual design is the requirement that the PE must be capable of operating in a totally distributed system environment to be applicable in a wide variety of distributed system architectures. With respect to communication facility characteristics and capabilities, this requirement precludes the assumptive reliance on a purely centralized or system-synchronous data transfer approach. Consequently, the PE should be able to operate in a totally asynchronous information transfer environment where data transfer events are not strictly time-ordered. This design approach is in partial contrast to present integrated avionics system design efforts, but it affords a more truly generalized, versatile approach which is compatible with a wide range of potential system applications. This inter-PE communications facility approach is vital in the realization of a truly versatile, system building-block PE design.

### 2. Aircraft Equipment Compatibility

To ease the integration effort needed to deploy DP/M in existing aircraft systems and thus enhance the overall cost effectiveness of the DP/M approach, it was considered desirable to attempt compatibility between the communications facility design and existing aircraft information distribution equipment. Standardization approaches to aircraft internal data distribution equipment are underway within the military. It is believed that these programs will result in a forthcoming generation of digital avionics devices with standardized interfaces to a data busing facility. The foremost representative of current programs is MIL-STD-1553.

For the purpose of future aircraft system compatibility, it was decided to provide DP/M compatibility with the design intent expressed for avionic system configurations in MIL-STD-1553, i.e., given the visibility afforded by this standard into the probable characteristics of future aircraft system equipment, the DP/M communication interface facilities design should reflect these characteristics. In particular, the characteristics under concern are those of digital aircraft equipment interfaces and physical data distribution facilities (e.g., data bus characteristics). Therefore, physical and electrical compatibility with equipments derived from MIL-STD-1553 is emphasized, whereas exact functional adherence to the standard is of second-order significance

## 3. System Performance Requirements

The DP/M PI must not only meet the functional requirements imposed by its distributed system environment, but must also be compatible with both the general and the unique performance requirements of its intended application(s). Accordingly, the following system communications requirements were identified and analyzed to define the performance characteristics required of the information transfer facility design.

### *a Bus Traffic Requirements*

It has been ascertained in several studies that the actual communications bandwidth requirements of contemporary avionic systems can be easily met with 1 Mbps serial data link. An analysis of the data transfer requirements of each processing task envisioned for the DP/M baseline system was performed using the data generated from previous DP/M and integrated avionic mission processing requirements definition studies (see bibliography). Because it was decided to maintain electrical interface compatibility with conventional aircraft data bus standards, a 1 Mbps serial bus data rate was assumed and all traffic estimates were thus measured in units of kilobits per second (Kbps).

The bus traffic analysis reflects the results of applying the busing system design to the data transfer requirements of the worst-case mission-processing segment. The actual amount of bus traffic encountered by the busing system at any given period in the life of an aircraft mission is dynamic (varies within predetermined limits), depending on the processing requirements of different mission segments. The segment analyzed (postulated as the worst-case processing segment, both in terms of processing throughput and bus communication traffic. Detailed information concerning busing protocol characteristics and additional overhead imposed by those characteristics is discussed in a subsequent section, for purposes relevant here, bus traffic is composed of transmission of unique serial data bit strings known as messages. Each message contains the basic generic information elements listed in Table 2.

These data distribution characteristics were used in determing the bus traffic decomposition and subsequent overhead contribution analysis. Each element of information transferred via a bus in order to move data from PI to PI was segregated to allow a determination of the relative amount of bus bandwidth consumed by that individual element function. This data was desired to allow (via busing facility design modification) elimination of overbearing overhead elements, if required.

The total message flow required between interacting processing tasks was segregated into global and local groupings. The overall system data distribution traffic requirements of the baseline system are summarized in Table 3. The global bus traffic is comprised of both intefunctional (inter-affinity group) data and Global Executive Command Control information. The bus traffic required for the executive functions was estimated from assumed function subfunction scheduling command requirements in keeping with the Global Executive Control strategy described in Section VIII of this report. Local bus traffic is a function of the physical interconnections of sets of PIs within affinity groups. In the local bus traffic analysis, it was assumed that PIs and their associated processing tasks are grouped into common functional groupings (e.g., navigation, weapon delivery, etc.). In addition, to arrive at a worst-case bus traffic prediction, each PI within an affinity group (functional grouping) was assumed to contain a single processing task involved in that function. In actuality, more than one task will be

TABLE 2. INFORMATION TRANSFER BUS TRAFFIC DECOMPOSITION

| Traffic Element | Global Traffic (Kbps/Percent of Total) | Local Traffic (Kbps/Percent of Total) | Total Combined Traffic (Kbps/Percent of Total) |
|---|---|---|---|
| Raw data | 33.6 49.0 | 234.1 68.9 | 267.7 65.6 |
| Data packing | 13.8 20.1 | 65.5 19.3 | 79.3 19.4 |
| Message header | 12.3 17.9 | 14.6 4.3 | 26.9 6.6 |
| Message sync | 2.0 2.9 | 2.2 0.7 | 4.2 1.0 |
| Data parity | 3.6 5.3 | 19.5 5.7 | 23.1 5.7 |
| Intermessage gap time | 3.3 4.8 | 3.7 1.1 | 7.0 1.7 |
| Total traffic | 68.6 Kbps | 339.6 Kbps | 408.2 Kbps |
| Efficiency | 49.0 percent | 68.9 percent | 65.6 percent |

TABLE 3. DATA DISTRIBUTION TRAFFIC (BANDWIDTH) REQUIREMENTS

| Data Type | Communication Facility | Traffic (kilobits/second) |
|---|---|---|
| Interfunctional | Global bus | 69.5 |
| System control (executive) | Global bus | 105.8 |
| Intrafunctional | Local bus | |
| Maximum function | | 171.5 |
| Total System | | 345.7 |
| External Device | Input output | |
| Maximum PI | | 522.2 |
| Total system | | 762.4 |

allocated per PE since the envisioned processing power (throughput) of a PE is substantially greater than that required for the performance of most tasks identified. Thus, the results shown for the local bus traffic represent a very conservative worst-case requirement.

It is recognized, however, that actual traffic encountered in a particular system configuration, may require addition of some I/O data into the busing system for reasons of physical distribution, thus causing the bus traffic to increase somewhat at either or both global and local levels. Assuming a 1 Mbps data rate capability for the information transfer buses, this concern is greatly relieved owing to the amount of bus bandwidth margin remaining and the fact that most distributed I/O data will be confined to affinity groups (local buses only). The bandwidth margin experienced indicates that most PEs of the assumed processing caliber used in an avionics processing suite will tend to be ome throughput or storage bound before they

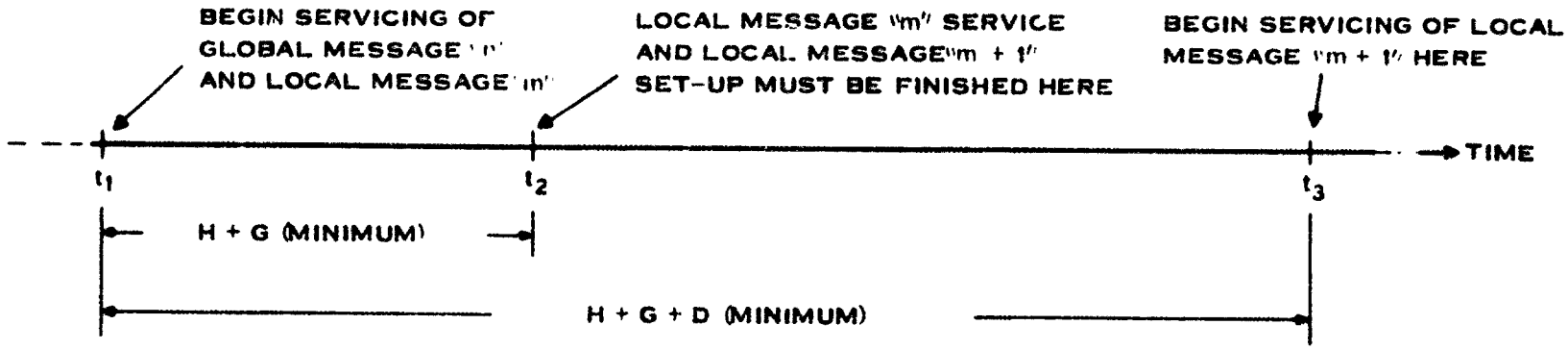
approach becoming "I/O-bound." This apparent inefficient use of the PE's functional resources is actually a desirable situation with respect to the DP/M system in that it indicates flexibility for future system growth/reconfiguration in the communication facility of the system building block, the PE. The communication facility is the one functional resource which typically restrains "expansion" as system growth is introduced; i.e., as functional capabilities are added to a DP/M system, the added processing load can usually be countered by the addition of PEs within the function, but the added communications traffic introduced by this addition must be absorbed by the existing busing facility. This situation exists because the bandwidth capabilities of the communication system are fixed at the time of design and are not incrementally expandable as is typically the case with processing and memory resources. Therefore, the bus traffic analysis indicates that the proposed communication facility will offer a comfortable degree of system expansion flexibility unless data transfer overhead, caused by the facilities protocol requirements, significantly increases total bus traffic.

### *b. Real-Time Response Requirements*

In addition to the data throughput requirement on the busing facility by inter-PE communications, the asynchronous nature of the distributed system requires that special consideration be given to the response time of the communications interface. The response time addressed here is the time allowed to process a received message in a given PE and to setup the bus-interface before reception of the subsequently received transmission. This control timing relationship is especially rigorous in the totally asynchronous distributed environment since the time interval between messages transmitted to, or received by, a given PE is not easily controllable within the system without sacrificing total system performance and/or interface hardware complexity. Therefore, the interface element and the communication protocol design were formulated, with this consideration being given priority. The actual response time constraint was analyzed in the following manner.

Bus message response time is primarily governed by the amount of time spent in processing input messages as they are received in real time. The process of input-message handling includes the actions required on the part of the Local Executive software (LEX) to properly recognize, validate, correlate, and disseminate the incoming messages for application tasks from both global and local bus interfaces. A worst-case analysis of response/reaction timing to input messages was performed, assuming the asynchronous busing facility operational characteristics.

Vectoring of the received message data sets to user-accessible buffer areas was recognized as an area of design concern. Timing considerations associated with message vectoring require that the bus interface be set-up to re-vector each successive message by establishing a new buffer starting address and buffer length to the bus interface input channel(s). This set-up process must occur before the receipt of the next message or the required data will be lost because of the "over-writing" effect of the following message input. To determine the severity of such response timing constraints, a worst-case message transfer situation was determined and analyzed. The situation identified is shown in Figure 7. It occurs when the PE receives both global and local input message completions simultaneously. Following receipt of these messages, if the next, immediate local message is a message destined to the same PE (i.e., back-to-back message receipt), the minimum time allowed before the input channel begins the next message input




EVENT TIMING:

AT $t_1$: GLOBAL MESSAGE 'n' AND LOCAL MESSAGE 'm' COMPLETE IN COMMON PE
$t_2$: NEXT LOCAL BUS MESSAGE ('M + 1') HEADER IS ASSEMBLED AND MATCHED IN PE
$t_3$: LOCAL MESSAGE 'm + 1' COMPLETES IN PE

TIMING CONSTRAINT

PE MUST BE CAPABLE OF SERVICING TWO MESSAGES IN $(t_2 - t_1)$ OR $(t_3 - t_1)$, DEPENDING ON BIU CONTROL REGISTER (MEMORY ADDRESS) BUFFERING FACILITIES

H = HEADER WORD LENGTH ( = 20 μS)
G = INTER-MESSAGE GAP TIME ( = 2 μS MINIMUM)
D = DATA WORD LENGTH ( = 20 μS)

Figure 7. Worst-Case Timing Analysis of PE Bus Message Servicing

(and, thus, both requires the new buffer set-up information and destroys the last received message identification) is only 42 microseconds. This response time is well beyond the interrupt response time and processing capabilities of a micro-class processor. It was determined that the Bus Interface Unit (BIU) design should aid and facilitate the solution of this response-time problem.

*c. PE Performance Aiding*

As mentioned in the overall objectives of the inter-PE communications facility design, this facility should not place any undue burden on processing requirements for the PE and should effectively augment the PE in its message processing duties by alleviating processor intervention in message input output activities as much as possible without severely complicating the bus interface hardware requirements. This design goal is very important in allowing maximum use of the PE in all information processing tasks due to a reduction in "overhead" activities.

*d. System Design Simplicity*

The information transfer facility should be designed to use simplified, low-risk, low-cost hardware. Special attention should be given to design impact on modular LSI implementation applicability. Additionally, the communication facility structure should allow use of homogeneous elements of as few unique part types as practical if the goals of a modular building-block approach to versatile system design and low life-cycle cost are to be eventually realized.

**4. System-Level Fault-Tolerance Implications**

Since the global busing facility is the only "central" resource of the DP/M system, it should be fault-tolerant within itself and should be instrumental in aiding fault recovery in all other system elements. In keeping with the simplified system hardware approach, fault tolerance in the communication facility is effected by orderly, system-software controlled recovery procedures in response to detected errors faults within the communications facility. Thus, it is the responsibility of the bus interface hardware to provide the necessary failure detection mechanisms required to accommodate this approach to fault tolerance at the system level.

## C. BUSING FACILITY DESIGN CHARACTERISTICS AND TRADEOFF CONSIDERATIONS

The present functional design of the inter-PE communication facility is the result of a series of design alternative and tradeoff investigations which progressed gradually during the study. Tradeoffs were judged in accordance with the system-level communications facility design criteria set forth in previous discussions. The following discussions are related to the delineation of various basic busing system functional design parameters and the design tradeoffs.

**1. Data Link Type**

All inter-PE communications are accomplished via a set of bit-serial, time-division multiplexed (TDM), 1 Mbps data-rate, twisted-pair cable (single signal line) bus data links. The recommended bus common and electrical interface characteristics of these serial information

transfer buses are identical to those defined and specified by recent standards for aircraft information distribution systems (e.g., MIL-STD-1553). The advantages and applicability of this approach to information distribution in the aircraft environment are documented in contemporary studies and need not be repeated here. In addition to the functional and physical advantages offered by this data distribution approach to the DP/M system, adoption of this standardized bus common hardware concept for DP/M would greatly facilitate integration of a DP/M system into future avionics systems incorporating standardized bus equipments.

### 2. Bus Language

Due to considerations of physical distribution and length disparities allowed in PE-to-PE bus lengths (up to 300 feet) and reliability in the aircraft environment, data should be conveyed along the bus using an asynchronous transmission technique. Asynchronous transmission means transmitting information in a manner which combines clocking, or bit-framing, information along with the actual binary data in a single signal.

To meet the above requirement, biphase level (Manchester II) binary data encoding is to be used for bus data representation. The encoding technique is popular in aircraft data bus standards (i.e., MIL-STD-1553); thus, it is a proven technique, ensuring DP/M communications facility compatibility with available (future) aircraft data bus hardware (Manchester II encode/decode equipment). The DP/M busing facility should allow use of the data and synchronization signal representation specified in MIL-STD-1553 and shown in Figure 8.

### 3. Bus Control

Since the physical inter-PE communications media are shared by multiple PEs, some mechanism must be included in the communication facility design to provide proper allocation control of the common bus facility among all users of the bus. Bus control means how the individual PE allowed to transmit data on a given bus at a given time is determined. Basically, there are two general approaches to achieving orderly bus control: "Centralized" and "Decentralized." The main difference between the two techniques is the physical location of the bus access resolving logic.

In general, a centralized control scheme requires that a single, "intelligent" control entity included in the system is responsible for all control activity. In a DP/M system, however, this philosophy has several disadvantages compared with a distributed, or decentralized approach. Since the DP/M system is composed of a set of homogeneous PEs, the hardware required to achieve centralized control must be used in each PE. Depending on the type of bus assignment technique used, total system hardware requirements may increase, which is equivalent to providing the hardware for decentralized control. Since the DP/M system is intended to accommodate multiple TDM buses (global plus local), a PE for each bus is needed to provide the control function. Depending on the allocation scheme used, this PE role of "Bus Executive" may use a significant portion of the processing capabilities of t e PEs responsible for bus control.

The centralized bus control hardware technique represents a critical, single-point system failure source because a fault in the control hardware could disable the entire system. Recovery from centralized controller failure requires complex redundancy or back-up controller scheme.

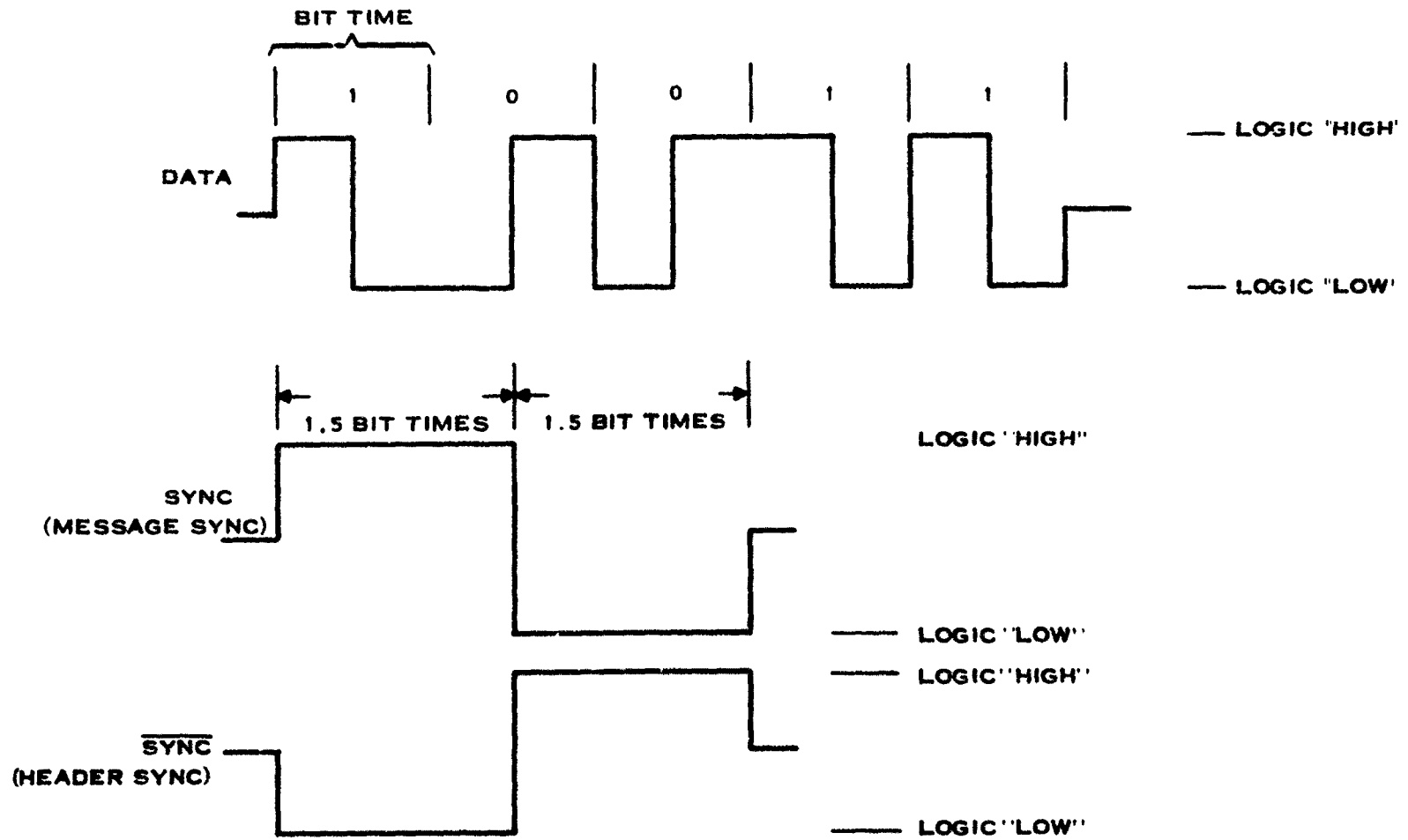



Figure 8. Manchester II Data/Synchronization Signal Waveforms

The reliability of properly implemented decentralized control is usually superior because most bus-control-related failures will only disconnect the associated PE from the bus. Depending on the allocation scheme used, the centralized control approach is usually less flexible for system growth or reconfiguration than is decentralized control.

Using a single bus line for transferring both allocation commands and data, a centralized control approach reduces effective bus bandwidth because of the bus overhead introduced by the presence of command information sent over the bus between each transfer of data (message transfer). Decentralized control results in faster allocation of the bus to users and offers more efficient use of available bus bandwidth. However, centralized control is better with respect to dynamic bus-user priority resolution and/or strict ordering or timing of the transfer of data elements (messages) with respect to each other. This advantage, though, is not of great significance in the avionics environment since the bus requirements of each PE (processing task) can be reasonably forecast at system design; i.e., the communication requirements are static, not dynamic in nature. Furthermore, the timing relationships between individual data transfers from (to) different processing tasks are typically not critical within the realm of responsiveness (though statistical) offered by a decentralized control technique with a 1 Mbps data rate bus. Exceptions to this general rule can be handled with proper bus assignment technique, as discussed subsequently.

Another potential advantage of centralized control is the hardware simplicity it affords when used with "dumb" terminals which do not possess the "intelligence" to determine their own bus access requirements. Again, however, this advantage is not applicable in a DP/M system where all bus terminals are "smart" terminals (PEs).

The above considerations of centralized and decentralized bus control attributes favor a decentralized approach in a DP/M system. A decentralized approach is further justified by consideration of the previously outlined DP/M system-level design objective which requires that the homogenous building block Processing Element be compatible with those system operational constraints present in a totally distributed system application environment. Centralized control in the context of a central command source for establishing and timing all inter-PE communications activity is representative of conventional "federated" system approaches. The federated system configuration is a special, but not general, class of distributed system architecture; and confining the communication facility to this class of environment was feared to preclude application of other possible system configurations. Therefore, the decentralized bus control approach was adopted for DP/M in an effort to eliminate any busing facility dependencies on particular distributed system applications or configurations.

## 4. Bus Assignment

Bus assignment refers to the method by which each bus user (PE) is allocated access to, or control of, the shared bus facility. Since the bus control technique favored was the decentralized approach, several basic types of bus allocation or assignment were considered; however, system-level desirability of using a single signal line (bus) for all data, timing, and control information reduces the set of candidate techniques, with the primary candidate being static scheduling according to "position" in a cyclic control sequence. A static scheduled bus assignment approach allocates the bus to individual users based on a repetitive, fixed-sequence algorithm. This technique is commonly used in TDM telemetry systems where the bus allocation requirements of each individual data source are predictable, based on *a priori* knowledge of

system operational characteristics. Each bus user, or data source, is assigned a unique "slot" in the total bus allocation procedure cycle.

The algorithm chosen for scheduling the bus facilities is a modified "round-robin" slotting technique which provides for simple advancing of the "bus control slot" from PE to PE, in a predetermined order, among the total set of PEs attached to the bus. This round-robin technique affords a modularity attribute typical of "daisy-chain" techniques without the reliability problems of daisy-chain control functions. This algorithm incurs a relatively unpredictable system responsiveness to user requests; i.e., the latency time between the generation of data (a message) to be transferred on a bus and the receiving of bus access by the associated PE is indeterminate. This latency time is a function of the number of bus users and messages transferred before control is procured. As a consequence of the avionics system and the DP/M system design philosophy, the avionics application can tolerate a statistical distribution of bus access latency without suffering degradation in performance. To accommodate unique or critical scheduling requirements of high-priority data transfers, it was determined that the bus assignment algorithm should allow some level of super-commutation of bus allocation time slots. The super-commutating approach allows assignment of multiple bus access slots, or usage opportunities, to particular bus users during each cycle through all users. The users can be prioritized to assure them of a particular percentage of available bus bandwidth and/or minimized access latency.

To advance or propagate the bus assignment sequence around the total set of bus users, a method was chosen for delimiting or framing individual bus transmissions (per allocation slot). Each bus transmission, referred to as a message, is terminated by the transmitting PE. Message termination is denoted by placing an end-of-message, or message-sync signal on the bus immediately following the last data bit in the message. If a PE cannot transmit data when it receives bus access, it merely "passes" control to the next subsequent bus-controlling PE by transmitting only the message-sync signal.

While a PE has control of a bus, it may transmit a variable-length message as desired. Considerations of response-time requirements for system control information transmission led to the decision to limit the maximum length of global bus messages to eight data words. This arbitrary choice was made as an attempt at a reasonable compromise between global message lengths encountered in the system processing requirements analysis and tentative estimates of Global Executive system-level response time (i.e., bus access latency) requirements.

## 5. Bus Communication Method

Given the above decisions to incorporate a decentralized control technique with static scheduling of bus allocation to multiple PE bus users, the next design characteristic addressed was the method by which a PE transmits data to other PEs after it has gained control of the bus. An asynchronous scheme of data transmission between transmitter and receiver(s) was already recognized as a firm requirement owing to consideration of physical distribution, data integrity, and overall reliability of the communication system. (The term "asynchronous" as used here means that no common data timing or clock source is required between transmitter and receiver.)

Asynchronous communication alternatives basically can be categorized into responsive and nonresponsive methods. Responsive communication requires a response from the data

destination, or receiver, which affirms the correct receipt of the transmission. This respond-back characteristic, however, increases bus traffic overhead and thus reduces effective bus bandwidth and efficiency. Principal advantages of the responsive technique are:

- Accommodation of communication between nonhomogeneous devices.
- Rapid notification to the data transmitter of correct or incorrect receipt of data at the receiver.

The former advantage obviously does not apply in a DP/M system because all bus users are homogeneous PEs. The latter advantage is valid but of uncertain value in the DP/M system application. Of primary importance, in conjunction with the receipt of incorrect data or a faulty transmission at a receiving PE, are detection of and notification of the fault to the data user (processing task). If transmitted data errors are detected in the message transmission, the user process has the basic options of:

Waiting for the next subsequent transmission of the data

Ignoring the data and using old or extrapolated data values from previous transmissions of the data set (applicable to iterative or cyclic processes)

If previous data cannot be used, a request can be sent to the data source for retransmission of the data set in error via normal bus access procedure; i.e., a responsive communication sequence can be effected with a nonresponsive facility.

If process or data criticality is such that none of the above basic options is satisfactory, the critical data (messages) may be multiply transmitted, with the transmitting PE assigned a high priority in the bus control sequence (super-commutated). Alternatively, critical data can be transmitted with error-correcting code techniques to allow reconstruction of the correct data by the data user. These procedures are applicable to offering a relatively high degree of responsiveness to message data correction requirements for data transmission errors only; message errors (data or message structure/protocol) caused by transmitter/receiver faults are typically nonrecoverable by a simple retransmit procedure. Thus, the degree of responsiveness in such cases is not critical since hardware fault recovery must be accomplished by either system software or hardware elements.

The preceding considerations resulted in the choice of a nonresponsive bus communication technique. The only advantageous feature of responsive communication, the facilitation of message retransmission due to detected message data errors, was not considered to be of special value in the DP/M system, especially when compared to the advantages of bus efficiency and hardware simplicity afforded by the nonresponsive technique.

The nonresponsive communication method chosen is an output demand driven method whereby each bus user transmits its exclusive data sets (messages) under its own cognizance after acquiring bus access. A possible alternative to this approach was to use a command-response technique whereby data can be either sent or requested by a transmitting PE. Accomplishment of real-time responsiveness to such data requests in a recipient PE would require either complex hardware in the bus interface unit and/or a decrease in effective bus bandwidth caused by data response latency.

Additionally, a data request scheme applied in the block-oriented, serial data transfer environment of the inter-PE communication system would result in detrimental effects on the applications processing throughput of the system because of applications-processing latency, or "hold-up," due to relatively lengthy data transfer times (i.e., the wait time a data-block user process must endure during the data retrieval and transmitting process). This data response latency time is a function of message overhead, the number of words in the data block, and the processing overhead time associated with handling the data in the recipient PE (interrupt routine time). This time was estimated to be on the order of 50 $\mu$s plus 17 $\mu$s for each word of data transferred. This magnitude of data response latency could easily increase effective applications processing time (hence, decreasing processing throughput) by a significant amount. Additional processing delay is possible because of round-robin bus access latency with respect to actual delivery of the data request to the data source via the associated information transfer bus. Possible latency contributions by this factor can significantly deteriorate overall processing throughput.

Possible alleviation of the above degrading effects on applications processing throughput may perhaps be accomplished by prudent Local Executive (LEX) software design (via "look-ahead" procedures, etc.). However, this solution was judged unsatisfactory because of the added LEX complexity it would require and the inescapable latency uncertainties associated with bus access delays.

Due to the above disadvantages associated with the performance of a "data request" mode of data transfer in the DP/M real-time environment and since it was decided that no real functional requirement for this mode of operation existed in the DP/M system application, it was decided that the system would operate in the fashion of an asynchronous, loosely ordered data sampling system with sample timing determined within each individual data source (PE). This mode of operation allows autonomous and asynchronous delivery of all user data before actual time of usage, foregoing the requirement of synchronous data request/response operations and their attendant real-time problems.

Once bus access is gained by a PE, it then transmits a message. The length of time that the PE is permitted to retain bus usage is variable and is equivalent to the duration of the single message transmitted. Each message is a word integral transmission of a variable number of data elements. Global messages can be optionally limited to a maximum of eight data words, enforced by system software protocol and monitored by bus interface hardware.

When a PE is not transmitting (i.e., at all times other than its bus access slot), it remains in an active "listening" mode as a potential receiver of each transmitted message.

### 6. Bus Synchronization

Data between a transmitter and all receivers can be synchronized at three basic levels: (1) bit, (2) word, and (3) message.

Bit synchronization is inherent in the encoded data format owing to the "embedded-clock" property of Manchester II modulation.

Word synchronization is not included in the message design since word synchronization can be effectively obtained by dividing the incoming data bits modulo the fixed data word length. In addition, no valid justification for including a word sync on the basis of data loss recovery during transmission errors has been identified. However, actual implementation of the BIU may impose word sync constraints on the incoming data stream within a message, such constraints, though, will not change the functional design of the busing facility.

An exception to this word sync philosophy exists for the special case of the message header which is the first word transmitted in a message. Since the integrity of this data word is critical to proper communication facility operation, it is delimited by an "end-of-header," or "header sync" synchronization signal to reliably delimit header information. This technique greatly improves the probability of detection of header data errors caused by bit timing errors.

Message synchronization is required to delimit the bus allocation time slots. Message sync denotes the end of a bus access slot and can thus be used by the distributed bus control logic to permit bus assignment sequencing at the proper time. The message sync chosen is an invalid Manchester II waveform so that synchronization information can be discriminated from data patterns.

To prevent the occurrence of message overlap caused by data propagation skew between widely separated PEs in the system, inter-message gap timing will be observed and enforced by each PE. Gap timing is a no-activity time inserted between the message sync signal and the beginning of the next message transmission. To ensure circumvention of data propagation skew problems in the DPM application where bus length diversity between PEs may be as great as 300 to 500 feet, a minimum gap time of 2-bit times, or 2 $\mu$s will be enforced. Each PE bus interface logic will wait a minimum of 2 $\mu$s after receiving bus control (via detection of a message sync signal) before beginning data transmission.

## 7. Message Structure

The technique used to establish transmitter receiver links for each message was a prime consideration in the conceptual design of the inter-PE busing facility. At the outset of the busing facility design, it was recognized that this single functional design characteristic would assert a prominent influence on the overall performance of the busing facility. Design of a message routing technique must consider its effects on the following system parameters.

- Bus bandwidth efficiency
- Processing throughput overhead incurred in address recognition
- Message handling in the message recipient
- System configuration and application flexibility
- Fault tolerance
- Hardware implementation requirements.

Accordingly, it was decided that the message routing, or addressing, technique adopted should:

- Efficiently support message broadcasts to multiple PEs

Incur minimum interference with ongoing PE processing during the address recognition process

Relieve PE processing responsibilities (software) associated with the handling and control of message receptions within the PE program

Accommodate system expansion and differing applications by eliminating physical dependencies or limitations to the fullest extent practical

Aid system reconfiguration in both on-line (in-flight) and off-line (system design) environments.

The routing approach chosen is a message-switched approach wherein all link-establishing information is present in the actual message transmission. Owing to the decentralized, static-scheduled bus assignment method used in the busing facility functional design, the only routing information required in each transmitted message is addressing information which can be interrogated by all other (nontransmitting) PEs to allow individual address recognition and ensuing message reception.

Conventional message routing/addressing schemes are based on specification of the physical address of the desired receiver. This approach, however, does not directly support message "broadcasting" to multiple receivers without complicating receiver hardware and increasing bus overhead. With this approach, broadcasting must be effected with the transmission of multiple headers to allow specification of all recipient PE physical addresses. Another disadvantage of the physical addressing approach exists when applied in a totally distributed system environment. As mentioned previously in the discussion of bus control design tradeoff considerations, the communication facility system-level design constraints precluded the assumption that a Central or Global Executive entity will exist in all DP/M applications. With the possible absence of this system control entity, it is not feasible to assume that the various distributed PE members of the system may each contain system "resource map" information if dynamic reconfiguration is required in system operation. Without this distributed knowledge, re[illegible]nment of messages to physical destinations is impossible, and therein lies another weakness of physical addressing in DP/M applications.

With the undesirable attributes of physical addressing identified, "logical" destination addressing was the next choice. Logical addressing requires assigning a physically independent identity, or logical address, to each bus data recipient. To achieve physical independence, identities are assigned to processes contained within each PE. Thus, the Bus Interface Unit within each PE must be capable of performing an associative address operation on each incoming message address with the set of process identities ("soft" names) resident within the PE. The number of unique associative "names" required per PE interface is thus dependent on the maximum number of processes allowable within a single PE. This number is indeterminate and must be arbitrarily chosen to accommodate a reasonable number of processes without overly complicating the hardware requirements of the PE Bus Interface Unit. To allow message broadcasts, a hierarchical addressing structure is available with the logical destination naming approach which allows a process to be addressed as a unique, single entity or as a member of a logical or functional group of processes. However, this technique limits process membership to a single group and does not allow a true universal, flexible broadcast capability without further complicating bus interface hardware.

During the design process associated with establishing a desirable message routing technique, it was determined that destination information alone did not provide enough information to allow correlation of a message's data content with a particular use or user; i.e., multiple data sets received by a given PE or a process within a PE must somehow be differentiated (message data demultiplexing). Message discrimination requires that a message identifier (Message I.D.) coexist in the message body with the actual data set being transferred. It was further recognized that this Message I.D. information is both necessary and sufficient for satisfying the functional requirements of a true broadcast routing approach, given that a complementary level of associative addressing capability exists in each potential recipient. Subsequent study of the possible exploitation of this unconventional message routing approach revealed it to be advantageous with respect to the DPSM inter-PE communications facility functional design objectives. This approach was adopted in the busing facility design. The detailed operational characteristics of this message routing/addressing technique appear subsequently, however, the functional advantages afforded by this approach are listed here for prefatory purposes:

Broadcast capability is inherent in the addressing method since messages are not directed to specific recipients by the transmitter (i.e., "source" addressing instead of "destination" addressing)

Message overhead is minimized since no destination addressing information is required in the message content, thus information transfer efficiency is increased

Message-routing determinations associated with the processes of system design and system configuration modification is simplified. Primary impact in this area is associated with the routing of messages from a data source (task) to all users. Usage of the Message Identity Associative Address (MIAA) routing scheme greatly simplifies routing assignments since no destination designation is required at the time of message generation. This concept is valid if each task is allowed to generate a single data set of all output variables generated within that task. Each user of any portion of the data set must then input the entire data set, possibly incurring a small amount of storage inefficiency. This mode of message generation and delivery is also desirable because it offers the greatest expediency in message delivery, requiring only one "pass" around the "bus control loop" to deliver a given set of data to all users as opposed to splitting a data set among various users which requires multiple messages and thus multiple "bus-loop" passes.

Correct and efficient message reception control within the PE is hardware facilitated, thereby removing some of the processing burden from the PE control software (Local Executive)

Data distribution is automatically managed by the associative addressing mechanism implemented in the bus interface hardware of each PE and therefore does not require allocation of PE processing to this function.

DPSM application in any system application where data distribution is determined by dynamic properties of the bus data traffic is permitted and facilitated.

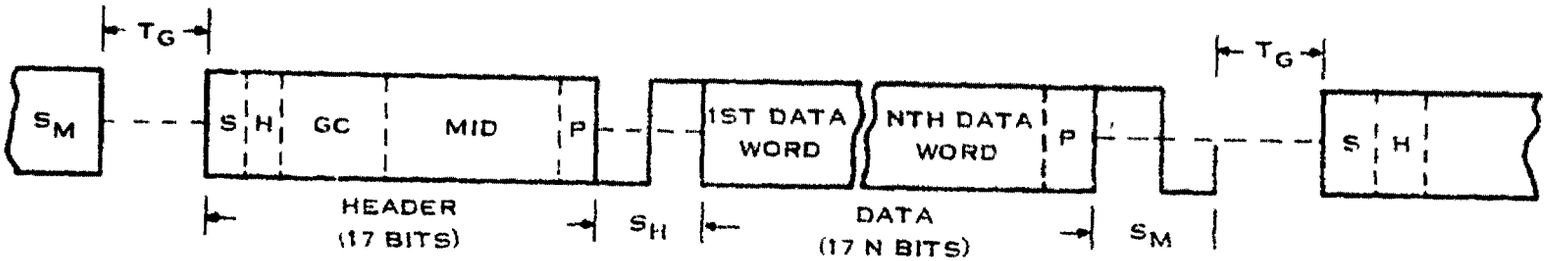
With proper implementation practices, the approach permits minimum bus interface hardware complexity associated with address recognition

The Message Identity Associative Addressing technique is a somewhat fundamental departure from the previously envisioned message routing scheme, but its impact on system design is advantageous. The scheme is an inherent broadcast message-addressing technique which associates the identity of the message data (or data source) with the message routing function. In effect, the technique operates by each processing task emitting an output data set with an associated identity header onto the bus and each potential receiver (all other bus users) actively checks the header (Message I.D.) to determine if the message is one which is to be received. Assuming that each PE has stored the "names" of all messages which it is interested in receiving (predetermined during system design), no destination routing information is required. The technique requires a single-address broadcast-only mode of bus message communication and uses the message format shown in Figure 9. This format allows accommodation of up to 1,024 unique messages (source data sets) in the DP/M system, an arbitrary choice based on compromising maximum (feasible) system message population requirements with minimum bus interface hardware complexity impact.

Each message is preceded by a header which contains all message routing and control information. The header length is chosen equal to the basic message data word length of 17 bits for reasons of hardware simplicity and functional capability. The message header is subdivided into the following control fields:

- **S**--The Message Start bit is fixed at a binary data "1" to denote the beginning of message transmission (required for Manchester II decoding operation).
- **H** - The High Priority Message tag is used to alert the message recipient(s) that the message requires immediate attention.
- **GC** - The 4-bit General Control field allows specification of special control/status information which further defines or alters the interpretations of the message data being transferred; field assignments are defined and interpreted by system software.
- **MID** The 10-bit Message Identity field specifies the identity of the associated message data. This field is used to establish desired transmitter/receiver links during message data transmission (It may also be interpreted as the data source address).
- **P** The Parity bit is used to force odd parity in the header data (17 bits) for data-validity checking purposes.

The message header is immediately succeeded by an end-of-header synchronization signal which uniquely delimits the header content from the remainder of the message data. The insertion of this Header Sync is intended to increase the probability of detecting header data errors with a simple parity technique, thus increasing system fault tolerance. Simple parity (one-bit) is included at the end of each word of a message. Previous statistical study has indicated that this level of error-checking code is more than sufficient to allow reliable detection of bus data errors. However, since the validity of header information is of particular importance to correct system operation, the Header Sync is used to reliably identify the parity bit location if data bits are added or dropped during header transmission/reception. Therefore, the Header Sync should be a reliably discriminated signal, unsusceptible to normal data bit errors. Consequently, a unique invalid Manchester bi-phase data pattern (shown in Figure 8) was chosen for this purpose. This signal is represented by a "long" low-level duration of 1.5 data bit times followed by a "long" high-level duration of 1.5 bit times, for a total sync signal duration of 3 equivalent data bit times.




$T_G$ = INTER MESSAGE GAP TIME (2-5 μS)

MESSAGE FORMAT DESCRIPTION:

| MESSAGE FIELD | FIELD LENGTH (DATA BITS) | FIELD FUNCTION |
|---|---|---|
| S | 1 | MESSAGE START CONTROL INDICATOR (=LOGIC ONE) |
| H | 1 | HIGH-PRIORITY MESSAGE TAG |
| GC | 4 | GENERAL-PURPOSE CONTROL FIELD (SOFTWARE DEFINABLE) |
| MID | 10 | MESSAGE DATA SET IDENTITY |
| P | 1 | PARITY BIT |
| SH | 3 | HEADER SYNC |
| SM | 3 | MESSAGE SYNC |
| DATA | 16 | DATA WORD |

Figure 9. Inter-PE Bus Message Structure/Format

The actual data content of the message immediately follows the header sync signal. Message transfers are block oriented; accordingly, the message data content is organized into fixed-length, 16-bit data words to match the PE word-length. A parity bit is added to each data word, resulting in an overall 17-bit message data word length. Each message data block is thus composed of an integral number of 17-bit data words. Each message may contain a variable number of data words. Global bus messages, however, can be optionally limited to a maximum length of 8 data words to guarantee global communication responsiveness by limiting global bus access latency.

It is of interest to note that no message length information is fixed into the message structure. In general, individual messages are nonvarying in length. The unique length of each message then is determined at system design and this information is "placed" in the user process (program). In exceptional circumstances, dynamic message length information can be inserted as a part of the message data content and handled by software convention.

Message transmission termination is denoted by an end-of-message synchronization appended immediately after the final data word. The occurrence of this Message Sync signal is used as the dynamic cueing mechanism which activates and controls sequencing of the distributed bus control logic throughout the DPCM system. If a PE has no data to transmit when given bus control, it can effect "passing" of bus control by emitting a "zero-length" message represented by an isolated Message Sync signal. Considerations of Message Sync validity requirements dictate the use of a unique invalid Manchester bi-phase signal as was the decision for the Header Sync signal. The Message Sync signal chosen is the binary complement of the Header Sync signal (as shown earlier in Figure 8).

## 8. Message Reception Control

The procedure known as Message Reception includes the primary activities of Message Recognition and Selection, and Message Vectoring which occur internal to the recipient PEs.

### *a. Message Recognition and Selection*

The message routing, or addressing, scheme chosen for the DPCM system essentially removes the responsibility of determining and specifying transmitter-receiver links from the transmitting PE to the receiving PE(s). Thus, some level of associative-match hardware must exist in the bus interface unit of each PE which is capable of storing the identity (or "address") of each message which is to be received. The message recognition logic actively scans each message header appearing on the bus in search of an address match.

As a matter of interest here, with respect to message addressing/routing design philosophy, it is important to note that any message reception capability within a receiving device in a multiplexed data system requires some type or level of associative memory implementation. The basic, intrinsic difference between the logically differentiated routing schemes which may be described as "physical destination," "logical destination," or "source" addressing is manifested in the amount of associative memory required for address-matching. For example, simple physical addressing effectively requires a one-word associative memory which contains the physical address of the receiver. Logical destination addressing usually requires the associative storage of a relatively small number of multiple logical identities which the receiver may assume. Depending

on implementation and/or the number of addresses allowed per receiver, source-level, or message identity, addressing may require an associative memory size equivalent to the total number of messages in existence. The immediate design implementation implication of such a scheme is that of relatively large and complex "associative memory" hardware requirements in the BIU. However, the novel implementation scheme proposed herein requires a relatively small amount of BIU hardware associated with input message recognition, less than that required for a logical destination name scheme.

Essentially, the method of associative addressing used "removes" the "associative memory" hardware normally used for input message response control (in the form of "name register" hardware) from the BIU into a "dedicated" portion of PE memory. This area is known as the Message Identity Associative Match Map and is equal in size (in bits) to the maximum number of unique messages required by a particular system application. The scheme associates each bit within the "associative-response map" with a particular message identity. The binary value of the associative bit determines message selection. The message population thus far experienced in the DP/M baseline system is in the 64 to 128 region. An arbitrary maximum message quantity is set at 1,024 messages, a choice which allows a comfortable system flexibility/growth margin while absorbing a relatively small percentage amount of the PE memory (1,024 bits → 64 words). However, it is important to note that the usage of the PE main memory is virtually "free" in a hardware economic sense, whereas design and implementation of similar associative memory functions within another unique LSI device (the BIU) is relatively expensive, both in terms of nonrecurring development and recurring hardware procurement costs.

Another consideration to be noted here is that the functional implementation proposed does somewhat degrade the processing throughput capability of the PE because of the memory cycle-stealing effects of the associative-match interrogation operations which occur coincidentally with the existence of each message header on both the Global and Local buses; however, an analysis of the baseline DP/M system message traffic indicates a memory bandwidth degradation of approximately 0.1 percent. Even reasonable worst-case estimates produce an anticipated maximum degradation of only an approximate 3 percent under improbable conditions. This analysis assumes a 1.0-$\mu$s memory cycle speed; faster memory cycle rates are considered possible for the DP/M PE, further reducing the magnitude of this already insignificant effect. Clearly, the practicality of the scheme is a function of the ratio of the bus bandwidth (bit rate) and resulting message rate versus the PE memory bandwidth. Present anticipated design parameters are well within the bounds of practicality for DP/M.

### *b. Message Vectoring*

"Message Vectoring" refers to the procedure by which an input message is handled within the PE as the message data is extracted from the bus. Basically, each message data set must be passed, or "vectored," to an area in PE memory where it can be stored for later use by an applications software process. Data buffering is required since the DP/M system operates in a loosely-coupled, asynchronous mode with respect to inter-process communication. Because of the data rate involved and PE processing overhead considerations in message transfers, to attempt handling of incoming messages on an interrupt-by-word basis is impractical. It was thus evident that an autonomous block-oriented Direct Memory Access (DMA), cycle-stealing capability must exist between the PE Bus Interface Unit (BIU) and the PE memory to minimize program processing intervention during input message data transfers. The method of autonomous DMA transfer control thus became a subject of design investigation.

Conventional autonomous block-oriented DMA data channels require processor (program) initialization, or "set-up" of the channel before beginning data transfers to memory. The initialization procedure allows the processor to specify control parameters for the impending transfer activity. The control parameters consist of a starting memory location which defines the data buffer beginning address and the number of words to be transferred, or buffer length. This initialization process is required per message received, thus associated processing overhead is directly dependent on input message volume and the amount of processing required to determine the desired transfer control parameters (i.e., the data buffer address and length).

In addition to processing overhead considerations, a more constraining performance requirement is placed on the input message handling capability owing to the nature of message traffic in a totally distributed environment. Essentially, a stringent message throughput requirement exists (internal to the PE) because of possible "back-to-back" message situations where multiple messages to be received by a single PE appear contiguously on the bus. This potentially troublesome situation is further compounded by the fact that the PE interfaces with two asynchronous buses (Global and Local) with potential simultaneous message activity occurring on each. From a worst-case message traffic condition analysis, it was determined that the minimum possible time between message completion in a recipient PE could be as small as 42 microseconds. This critically short allowable processing response time is further reduced to an effective 21 microseconds in the potential situation where processing service is required for simultaneously completed Global and Local messages with a subsequent, contiguous local message impending.

The above observations and considerations indicated the desirability of special BIU functional hardware features to solve the problem unless the hardware complexity incurred became prohibitive. Several schemes were formulated and considered. The initially considered hardware alternative involved, functionally simple double-buffering of the BIU input channel buffer address and length registers. This approach, however, allows an input message processing response time equal to only the length of the subsequently arriving message; thus, unless a minimum message length of 10 to 12 words is somehow enforced in the system, the result is not satisfactory.

The approach finally decided upon as a viable solution to the message processing response time problem uses a hardware-determined message-vectoring technique. "Hardware message vectoring" describes the process by which the Bus Interface Unit extracts or derives a buffer storage area of a message in PE memory from information existent in the message being received and then autonomously transfers the message data content to this area. No program intervention is required in setting up the input channel before each transfer. The functional implementation of the technique also allows the BIU hardware to validate the length of the input message with its expected (correct) length, "known" within the PE. To facilitate message processing subsequent to the buffering operation, an additional functional capability was incorporated within the BIU which permits posting of each received message identity in a first-in/first-out (FIFO) queue located in PE memory. This capability greatly reduces Local Executive (LEX) processing required to determine received message identities ex post facto. The functional processes involved are described in detail in subsequent section relating to the detailed functional design of the BIU. The overall advantages associated with the adoption of this technique in the DP/M system are several. The primary advantages are listed below:

Eliminates LEX message processing response time problem--no interface set-up operations are required by the program, and received message identification

queue posting is automatically maintained by the BIU. If double buffering is required, the LEX control software has the entire sample period of the data being double buffered to set up the alternate Buffer Address in memory.

Reduces LEX overhead significantly - the BIU hardware automatic message vectoring performs the overhead functions which otherwise would be handled by the LEX. This overhead corresponds to a significant reduction in LEX memory requirements (≈100 words of storage estimated); execution time savings are also appreciable. This results in increased applications task processing throughput capability within each PE.

Increases efficiency of buffer storage space - alternative dynamic buffer allocation schemes associated with LEX message buffer handling are wasteful of memory in that each buffer size must be of a quantum size equivalent to the longest message received by the PE. The Hardware Message Vectoring technique incurs no buffer size inefficiency.

Message length validity checking is eliminated from LEX responsibility, thereby resulting in additional software overhead savings.

Hardware Message Vectoring technique requires less BIU hardware for implementation than any other alternative scheme investigated.

The operation of this technique caused initial concern with respect to memory addressing validity and write-protect considerations since externally received data is used directly to determine an associated memory address. This concern was eliminated owing to the address validity assurance afforded by the message "address" recognition mechanism. The message selection mechanism inherently prevents undetected data error in non-selected message identities from becoming an agent in erroneous memory address derivation. The memory addresses derived and accessed by the BIU during message data input will correspond to correct software-determined message input control or buffer locations such that the accessing of random data in memory to be used as buffer control information is prevented.

## 9. Fault-Tolerance Considerations

Since the integrity of various system processing functions is dependent upon the correct transmittal of data between processes, the inter-PE communication facility should provide the capability to monitor all data communications between system PEs and alert the system upon the detection of data errors/faults. Since the Global communication facility (Global bus) is the only "centralized" system resource, it should be fault-tolerant within itself.

DP'M System fault-tolerance requirements, as applied to the information transfer busing facility, necessitate the existence of certain fault-detection mechanisms within the BIU of each PE to facilitate system fault-tolerance via fault detection/isolation and ensuing system software recovery techniques. The following discussions concern each unique fault detection capability to be implemented within the BIU design and the fault condition requiring the existence of each mechanism.

### *a. Bus Message/Data Errors*

Bus data errors may be caused by various phenomena: statistical signal-to-noise ratio induced error rate, spurious noise bursts, transmitter/receiver malfunction, etc. Such errors may

be evidenced by various manifestations in the bus data stream. With the bi-phase-encoded asynchronous data transmission technique used, such errors may be detected as: (1) improperly encoded data representations, (2) missing or added data bits, or (3) data content error (bit value error). The BIU will detect any of the above error manifestations in a received message data stream with the following mechanisms to ensure input data content validity:

Biphase encoding patterns received in the input/receiver section of the BIU are internally compared against legal Manchester II encoding patterns. This includes both data and unique synchronization signal representations.

Information content validity is checked, using simple odd-parity checking of each 16-bit word received (i.e., each data word transmitted on a bus is received as a 17-bit quantity in each receiver). Parity information is accumulated during the first 16 bits of each received data word and is compared against the 17th bit of each data word.

Information bit presence validity is checked subsequent to the reception of an entire message data stream. The check is performed by testing the received bit-string length for a modulo-17 length (i.e., an integral number of 17-bit words). This validity checking measure decreases the probability of nondetected data errors from that achieved by parity checking only.

Bus data error detection of the types described above are performed on each message word (i.e., both header and data). Header data errors are discriminated from data word errors in conjunction with BIU activity following error detection. If any data error is discovered in the content of the message header, the message is totally discarded as a possible candidate for reception; data errors detected during reception of the "data" content of a message do not change the normal input sequence; i.e., no message data will be "thrown-away" if a message data error is detected earlier in the message reception. It is intended that any message "dumping" be left to the discretion and option of the Local Executive Control (LEX) or the applications programs (data users) only.

*b. Message Protocol Errors*

Owing to the deterministic nature of avionics real-time processing requirements, all message definition and routing requirements can be determined off-line during system design (e.g., during Process Construction). Therefore, the amount of data content of each message is known *a priori* and this knowledge can be inserted into the IE program responsible for message reception control/management. The BIU automatically extracts this message length descriptor for each message received and verifies the correctness of the incoming data word count (message length). If a discrepancy between the expected message length and the received word-stream length is detected, the message is terminated at the shorter length.

A special message-length protocol is observed for all messages routed via the Global bus. All Global messages are required to be limited to a maximum data content of eight words to ensure a relatively quick system response-time (timeliness) for Global Executive control information transmission and control action initiation. Hardware enforcement of this protocol is provided in the BIU by detecting and inhibiting an attempted violation. Thus, an attempt by the PE operational program to transmit a message longer than eight words will result in hardware detection and the message transmission is aborted (i.e., no transmission is initiated) with an

appropriate interrupt stimulus generated to notify the processor (program) of the error. Apparent in this design, with the safeguard mechanisms above, is the attempt to minimize the probability of software-error-induced Global message length protocol violations in any PE (perhaps a "mad" PE) which may severely affect entire system operation.

The PE is notified of each of the above error occurrences via hardware flags set by each detection mechanism. Under program control, these flags can either by interrogated (read) or allowed to generate an appropriate interrupt stimulus, depending upon program option.

*c. Bus Control Errors/Faults*

Since the DP/M bus communications facility incorporates a distributed bus control scheme, each PE in the bus control "chain" or "loop" must actively participate during its designated access slot (time slot) to maintain proper bus control circulation throughout the DP/M system. Bus control faults can occur in two basic modes: bus quiescence--no access activity; or bus dominance--monopolizing or incessant activity.

A bus quiescence condition may be caused by "functional" or "physical" discontinuities in the bus control chain. A functional discontinuity is attributed to a faulty PE (for internal reason) which fails to participate (transmit) during its allocated time slot. Physical discontinuities are caused by failures in the bus common itself (e.g., broken cable, faulty connection, etc.)

To allow recovery from a bus quiescence fault, the condition must first be detected. Detection is provided in the BIU for both Global and Local buses by a Bus Quiescence Watchdog Timer (BQWT) implemented in each bus input interface section. The BQWT measures the duration of "no bus activity" periods (i.e., the absence of bus signal transitions, or data bit-clock). If a "no activity" period in excess of a maximum allowable quiescence time value is detected, the BQWT causes an appropriate interrupt stimulus to be generated and forces the Global output access mechanism (including the Bus Control activity) into a disabled state. The latter event is performed to bring the Communication System to a known state (halted) in preparation for ensuing recovery activity which is performed under the control of a system software recovery monitor. The BQWT time period is hardware-fixed at a period of time which is a function of anticipated bus control response delays in the BIU, bus path propagation delays, system initialization related timing considerations, and error detection response time requirements.

A bus dominance condition typically can be caused by a faulty BIU output interface section which acquires access of the bus, not necessarily during its appropriate allocation time slot, and retains access with incessant transmissions (e.g., a "chattering" PE). Hardware safeguards are provided in the BIU, as described previously in Subsection III.C.9.b to prevent dominating conditions caused by PE program execution errors. In the improbable event that these hardware safeguard mechanisms fail or the BIU output section (transmitter) fails in a "chattering" mode, the BIU provides a hardware Bus Dominance Watchdog Timer (BDWT) which essentially monitors the length of each independent bus access period, checking for a maximum length message violation. The BDWT may be implemented quite economically using the basic bit-count logic incorporated in the BIU message input section for input message assembly and length verification.

If the BDWT detects a maximum message length violation, an appropriate interrupt stimulus is generated and the BIU Output Control Section (including all Bus Control functions) is forced into a disabled state. This action forces the bus communication facility into a known state to facilitate recovery actions by a software routine and forces (i.e., attempts) termination of the faulty transmitter.

Multiple bus user access errors present a particularly troublesome problem in detection. This class of bus control faults/errors can be caused by either hardware or software faults/errors. Hardware faults are expected to typically result in an "out-of-position" PE which attempts to acquire bus access in another PE allocation slot. The result is indeterminate data presented on the bus, depending on the electrical coupling characteristic of the bus interface receiver/driver circuitry used. This case of multiple access errors will typically produce an eventually detected bus quiescent condition since an "out-of-order" PE will not honor its allocated time slot when reached in the system bus control cycle.

Unfortunately, unique failure modes can arise wherein the quiescent condition is not introduced, and thus non-detected. These specific situations occur when a PE assumes an erroneous bus position (or access slot period) which is a binary submultiple of its correct position in the bus control chain. For example, an incorrect position of "4" will overlap a correct position of "8." A more extreme example is the situation where a PE erroneously assumes a position value of "1" which causes the PE to transmit during every bus access slot. Direct, unambiguous detection of these particular multiple bus access situations is virtually impossible, but indirect detection can be accomplished via the recognition of other message/data errors which will be caused by the multiple access contention; i.e., multiple accesses will invariably result in continual data errors (bad parity and/or biphase encoding), and/or message length error, and/or Bus Dominance error caused by non-discriminated (non-valid) message sync signal reception. The latter of these indirect error manifestations will bring the bus communications facility to a halt. The data/message error manifestations can provide valuable information to the GEX (after a certain repetitious threshold is surpassed) about the fault's identity. This information can indicate that a dominant condition exists.

### *d. Bus Interface Fault Detection*

The bus interface unit is designed to allow a full-duplex mode of operation to accommodate self-test message transmission and simultaneous reception. This "rebound" test capability can be exploited under program control to verify correct bus interface operation at the small expense of added bus traffic.

### *e. Fault Detection Mechanism Control*

All of the hardware fault detection mechanisms included in the BIU design are under the control of PE program execution. All data/message error detections are recognized by the program at its option, depending on the associated interrupt mask setting associated with the fault/error-denoting interrupt stimulus. Those hardware detection mechanisms which affect BIU status (i.e., the BQWT, BDWT, and Global Output length limiting mechanisms) are also controlled (enabled/disabled) by the setting of their respective interrupt mask bit. Thus, it is imperative that the software commands/routines which control these devices be structured with fail-safe interlocks (internal pass word access codes, etc.) which prevent unauthorized or erroneous control of these mechanisms.

### *f. Redundant Global Buses*

It has been previously recognized that since the DP/M system Global bus is the only "central" system resource, Global fault-tolerance should be especially emphasized. It is also recognized that either catastrophic Global bus failure (i.e., an irreparable failure/fault which makes the Global bus useless) or certain "removable" faults require an alternate, redundant Global communication path to achieve system recovery. Potential inter-PE data busing failure modes have previously been identified and discussed. Appropriate provisions have been made in the BIU hardware to allow detection of related failures and consequently alert the system (via the LEX) of their occurrence so that fail-safe or failure recovery procedures can be performed. One primary important fault-tolerance design aspect deserves expanded attention with respect to busing facility design. This area of concern is Global bus back-up capability if the Global bus facility fails irreparably. Irreparable bus failure can occur as a result of physical failure/damage in the communication path itself (e.g., electrical interface devices, connectors, cable), or particular hardware/software failures within bus users can effect a bus monopolization condition and make the bus useless.

The above class of bus failures cannot be tolerated by the system unless some facility is provided to allow "circumvention" of the problem in order to allow fault-tolerant activities to occur. Clearly, this additional facility concept is required at the Global bus level to prevent a single bus-failure at that level from becoming "catastrophic" to the system. Global bus backup, or redundancy, is then considered as a primary fault-tolerant design objective within the busing facility. Global bus failure recovery via switching to an alternate bus can be effected in different manners. For purposes of hardware simplicity, the originally investigated scheme allowed for constructing a "quasi-global" bus from the set of local buses of the system as shown in Figure 10. If the global bus fails, the local bus nodes in each PE are switched to form a "daisy-chain" bus link connecting each PE, thereby effecting a "quasi-global" bus. However, two distinct disadvantages were discovered associated with this approach:

- Global bus failure recovery must be accomplished via "back door" (local buses) reconfiguration control information passage under the direct control of each and every PE; therefore, the construction of a quasi-global chain of local buses requires that all PEs in the system be "well;" i.e., a failed PE which cannot perform the necessary local bus node switching (for whatever reason) will prevent the quasi-global configuration from occurring. The failed PE may have been the cause of the failed global bus.
- Global bus failure recovery may force degraded system operation because of combined Global/Local bus traffic of entire system onto a single bus facility. (The dual-level busing structure disappears.)

During the analysis of the above global bus backup concept, it became apparent that an alternate, or redundant, facility of some type must be incorporated in the busing facility design to allow global bus reconfiguration control information passage to all PEs if a primary global bus failure is detected. Clearly, the operation of the facility should not depend upon or be affected by the operational integrity of any individual system element (PE). The intuitive concept of redundant global buses was then introduced for investigation and tradeoff analysis.

The simple redundant bus approach is shown in Figure 11. This approach allows for the routing of doubly redundant, identical global bus lines to every PE in the system. This approach

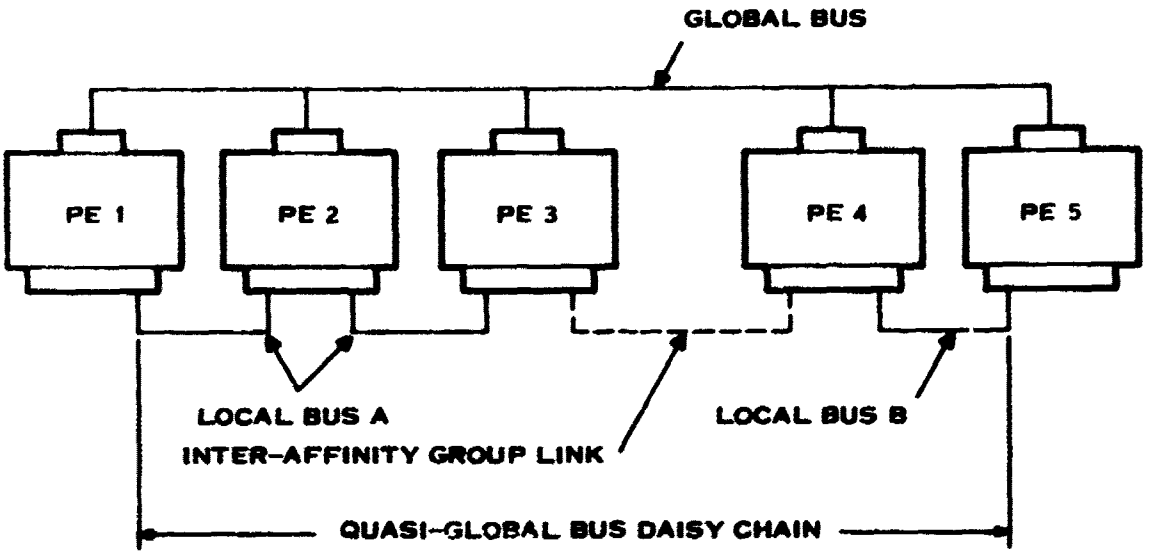



**Figure 10. Quasi-Global Bus Configuration**

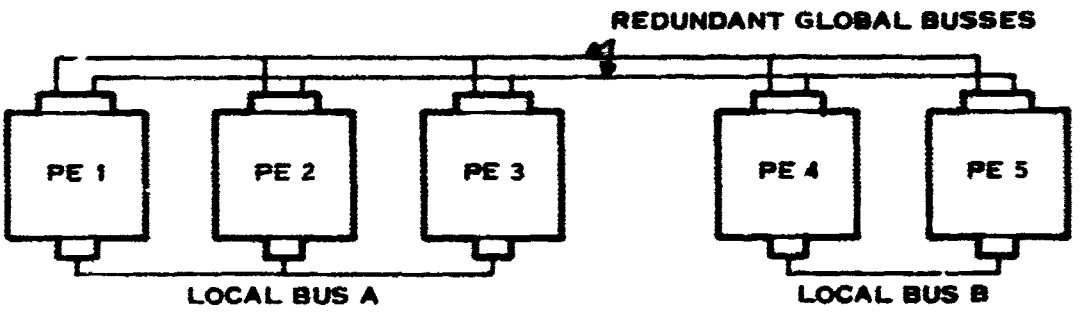



**Figure 11. Redundant Global Bus Configuration**

requires an increase in system cabling over the previous scheme, but the increase is insignificant since the additional cabling is in the form of twisted, shielded-pair transmission line, and the previous scheme required additional cabling between affinity group clusters (effectively "pieces" of a redundant global bus). Design questions to be answered with respect to this approach include the technique of alternate bus usage, or switchover, control and Global Bus Interface functional element redundancy requirements.

Functional redundancy desirability versus BIU hardware complexity tradeoffs resulted in the design choice of a single Global Bus Interface functional hardware element with the capability of switchable interfacing with redundant physical global bus paths. Switchover control requires hardware augmentation to allow establishment of the proper bus usage. The underlying philosophy of the sought approach is to provide backup, or failure recovery, capability for the global bus communications path only, not to provide each PE with Global Bus Interface failure-recovery at its local level [i.e., failure recovery techniques are required at the system level only, not at the individual resource (PE) level].

The above mentioned Global bus redundancy philosophy led to the search for a functional redundancy design approach which would offer satisfactory Global bus reliability while maintaining abbreviated hardware complexity additions in the BIU. As a first step toward formulating such an approach, the responsibilities of a Global bus redundancy management function section of the BIU were identified to be:

Provide a redundant bus "listening" capability to detect alternate bus activity (fault recovery commands)

Detect a "switchover" situation

Select the alternate Global bus as the valid input source and switch the Global reception section to the alternate (backup) bus

Provide switching of the Global interface output to either of the redundant Global bus paths under software control.

The redundant Global bus management scheme adopted has resulted from the aforementioned functional design considerations as well as a somewhat speculative but conservative future LSI implementation impact consideration. The approach is represented in the functional block diagram of Figure 12. The primary underlying system level fault-tolerance assumption reflected in this approach is that all Global bus failure recovery activity will be initiated and supervised by an intelligent software system entity, the Global Executive (GEX) [i.e., the detection of a failure condition requiring use of the backup bus and the ensuing system recovery commands (via the backup bus) will be left to the discretion of the GEX software only]. The redundancy management logic performs the following activities involved in "catastrophic" primary Global bus failure recovery.

When a recovery action is decided upon by the GEX, it issues a "switch to alternate bus" command (predefined, dedicated message identity) over the alternate bus. Transmission access to the alternate bus by the GEX PE is effected by previously commanding the redundancy management logic to switch the Global bus interface transmitter to the alternate bus with a "Switch Global Output" I/O instruction. Upon reception of a valid "switch to alternate bus" command message on the alternate (standby) Global bus, the redundancy management logic in each PE BIU (including the GEX PE) will automatically switch (connect) the Global bus input interface to the alternate (now primary) bus, and all subsequent communications will be performed via the new bus connection. This input switchover action will terminate any current message input activity in each PE bus interface by forcing a "reset" of the associated input interface control logic.

Now that the GEX has the system "listening" to the backup bus, it can transmit recovery action commands in a normal Global communication mode. The procedure followed for recovery

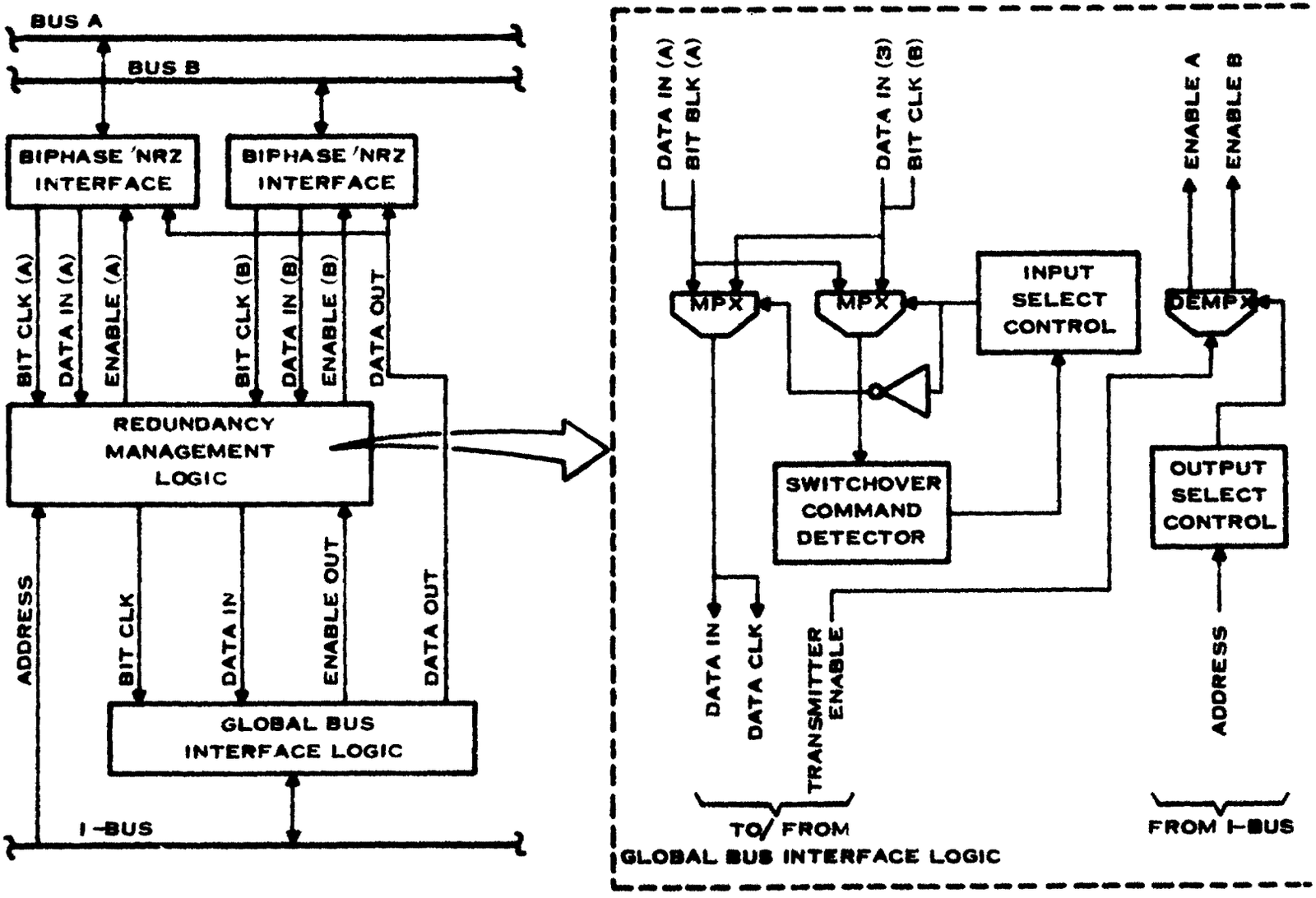



Figure 12. Global Bus Redundancy Management Functional Approach

will be similar to a system initialization control sequence. Depending on the detected nature of the failure, each PF may be individually commanded to connect its output to the newly appointed primary bus for diagnosis purposes, or all PFs can be "switched" in unison with a single command communicated via the newly appointed Global bus.

Each time a bus switchover activity is initiated, the roles of the two physical Global buses are reversed (i.e., "primary" or "alternate"). Therefore, during a Global bus failure recovery procedure, the bus configuration can be switched more than once, if required. This capability is intended to prevent "lockup" of the backup bus if a "mad" PF which caused the primary bus failure is switched onto the backup bus during the recovery process. When this occurs, the GEX may force switchover to the previous configuration with the failure now removed to the "standby" bus. This approach will accommodate all modes of single failures in the Global bus facility.

The Global bus redundancy management scheme described above appears satisfactory in meeting Global bus reliability goals with a viable, low-hardware-complexity implementation in the BIU.

## D. BUS INTERFACE UNIT FUNCTIONAL DESIGN DESCRIPTION

The Bus Interface Unit (BIU) provides the hardware elements required to implement the bus communication facility design requirements and concepts previously described. The major functional elements implemented within the baseline BIU design are shown in Figure 13. Owing to consideration of actual functional and performance characteristics required of the BIU at both the Global and Local levels, and the desirability of hardware simplicity/commonality within the BIU, it was recognized that the same functional elements could be replicated for both Global and Local interfaces, with only minor differences which will be identified in the following description of each functional partition of the BIU.

### 1. Bus Data Translation

The Bus Data Translation functional element is responsible for providing interface compatibility between the biphase-encoded serial data transmission bus facility and the Non Return to Zero (NRZ) parallel data orientated PF. This functional element also performs integrity checks on the incoming serial data. The Bus Data Translation functional element is composed of two major subfunctional elements described below.

#### *a. Biphase Data Decode/Assembly/Verification*

This subfunctional element of the BIU decodes the bi-phase-encoded bus data into binary NRZ data and bit-clock information. The incoming serial data stream is converted into parallel 16-bit word format for transmission to the PF memory. The incoming data is also scanned for recognition of the unique synchronization patterns and notifies the Bus Input Channel Control and Message Reception Control functional elements of their occurrence. Also performed in this section is input data validity checking which verifies encoded data format, data word length, and data word parity correctness. Detected errors are notified to the Bus Input Channel Control functional element.

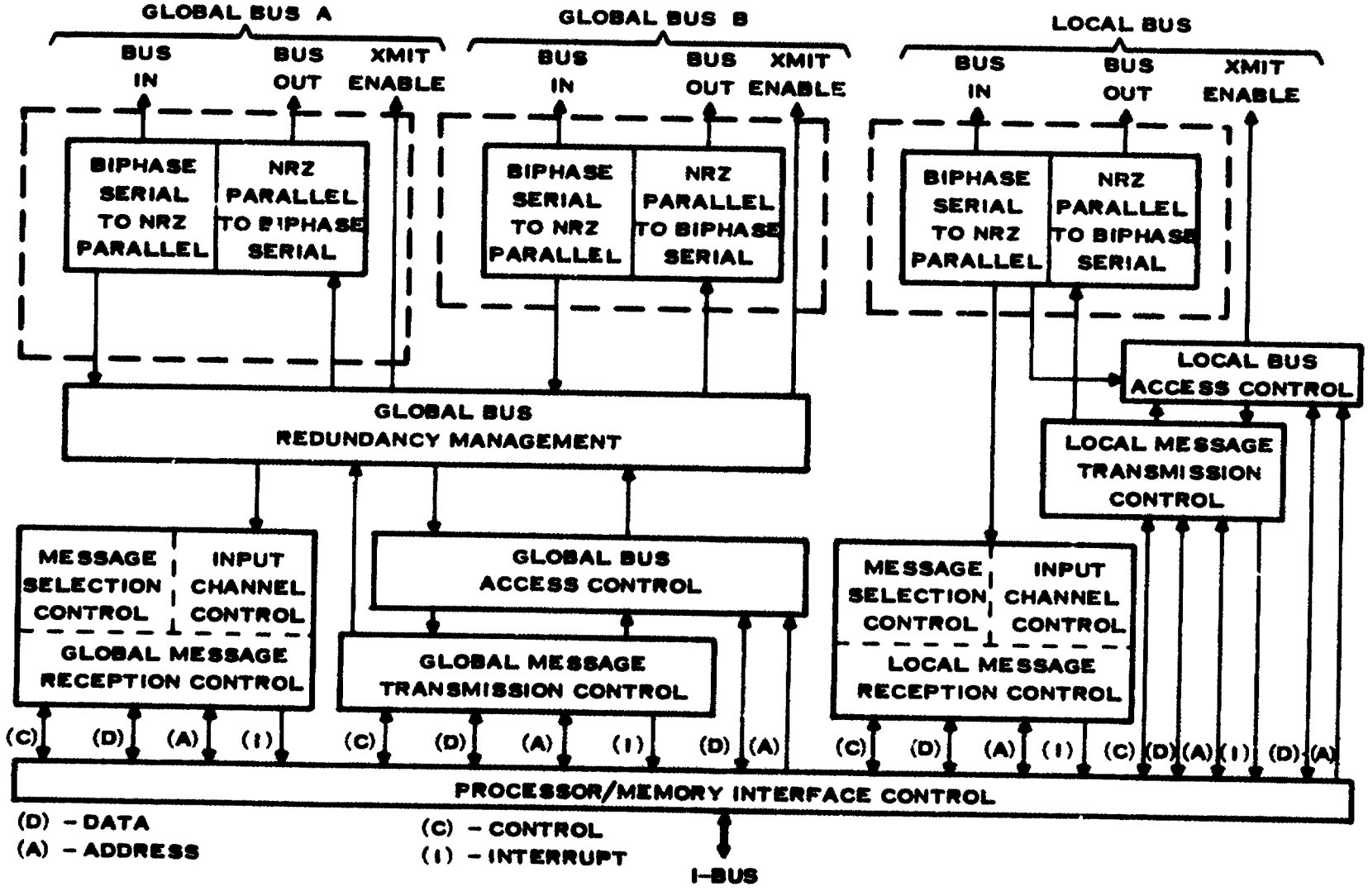



Figure 13. BIU Functional Block Diagram

*b. Biphase Data Encode/Transmit*

This subfunctional element converts the parallel 16-bit data words extracted from the PE memory by the Output Channel Control Functional element into a serial bit-stream of bi-phase-encoded data transmitted onto the bus. Parity is generated for each data word and inserted as the last bit in a 17-bit transmitted word. This section also generates the unique bi-phase synchronization signals as commanded by the Bus Access Control functional element (for "passing" bus control) or the Bus Output Channel Control functional element (for terminating data transmission or header denoting).

2. **Message Reception Control**

The message reception control functional element is composed of two primary functional sub-elements, Message Selection Control and Bus Input Channel Control.

*a. Message Selection Control*

The Message Selection Control functional element interrogates each message header as it is accumulated from the bus in the Bus Data Translation functional element. This element "maps" the message header (which is the Message Identity) into a physical PE address within the Message Input Associative Match Map (MIAMM) space in PE memory (refer to the procedure shown in Figure 14). The control logic then extracts the word addressed in MIAMM and tests the proper associative response bit associated with the received header (i.e., the Message I.D. is mapped into a physical bit address in PE memory). Detection of a "set" bit (logic "1") indicates that the message is to be received by the PE; detection of a "reset" bit (logic "0") indicates that the message is not relevant to the PE, and thus, ensuing message data should not be input. The correct associative match map information is assumed to have been placed in the MIAMM memory location at time of program initialization. Positive detection (recognition) is relayed to the Bus Input Channel Control functional element to initiate the data input sequence.

*b. Bus Input Channel Control*

The Bus Input Channel Control functional element assumes responsibility for message data input to the PE memory subsequent to notification of input enable detection from the Message Selection Control section. The procedure by which message input is achieved is shown in Figure 15 and follows the sequence below:

The message header now resident in the Input Data Register (IDR) is input to the PE memory location currently addressed by the Input Queue Pointer (IQP). The identity of the received message is thus placed into a FIFO (first-in-first-out) queue of modulo (8) words in a dedicated portion of PE memory. This action provides automatic hardware queue posting of input message identities to be subsequently scanned and processed by the processor Local Executive software (LEX) at its convenience and throughput capability, thus eliminating interrupt response execution time overhead, which would be incurred subsequent to each individual message received. (The IQP is later incremented at message termination.)

The message header in the IDR is now "mapped" into a physical PE memory address corresponding to its appropriate location in a pseudo-dedicated

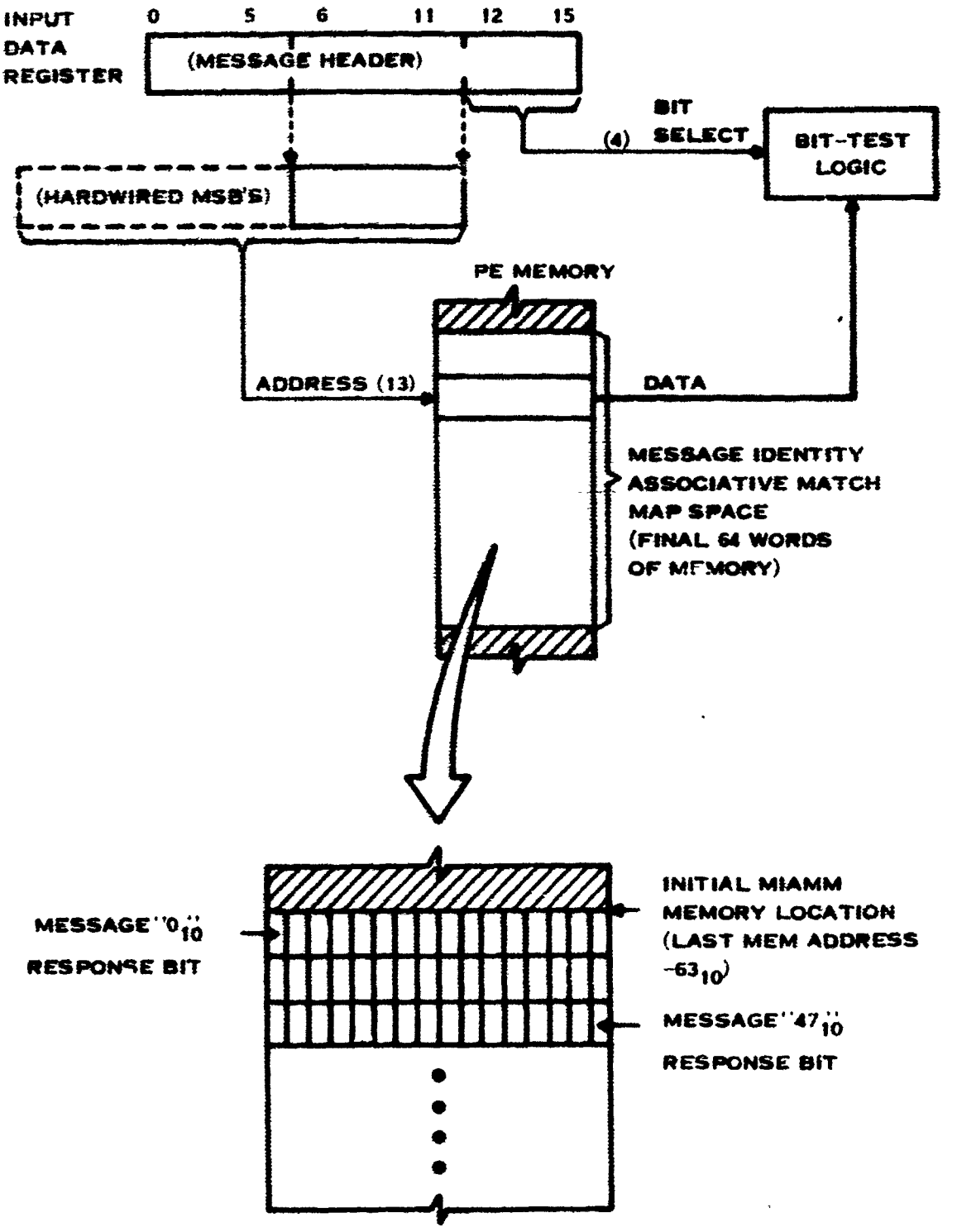



Figure 14. Message Identity Associative Addressing Operation

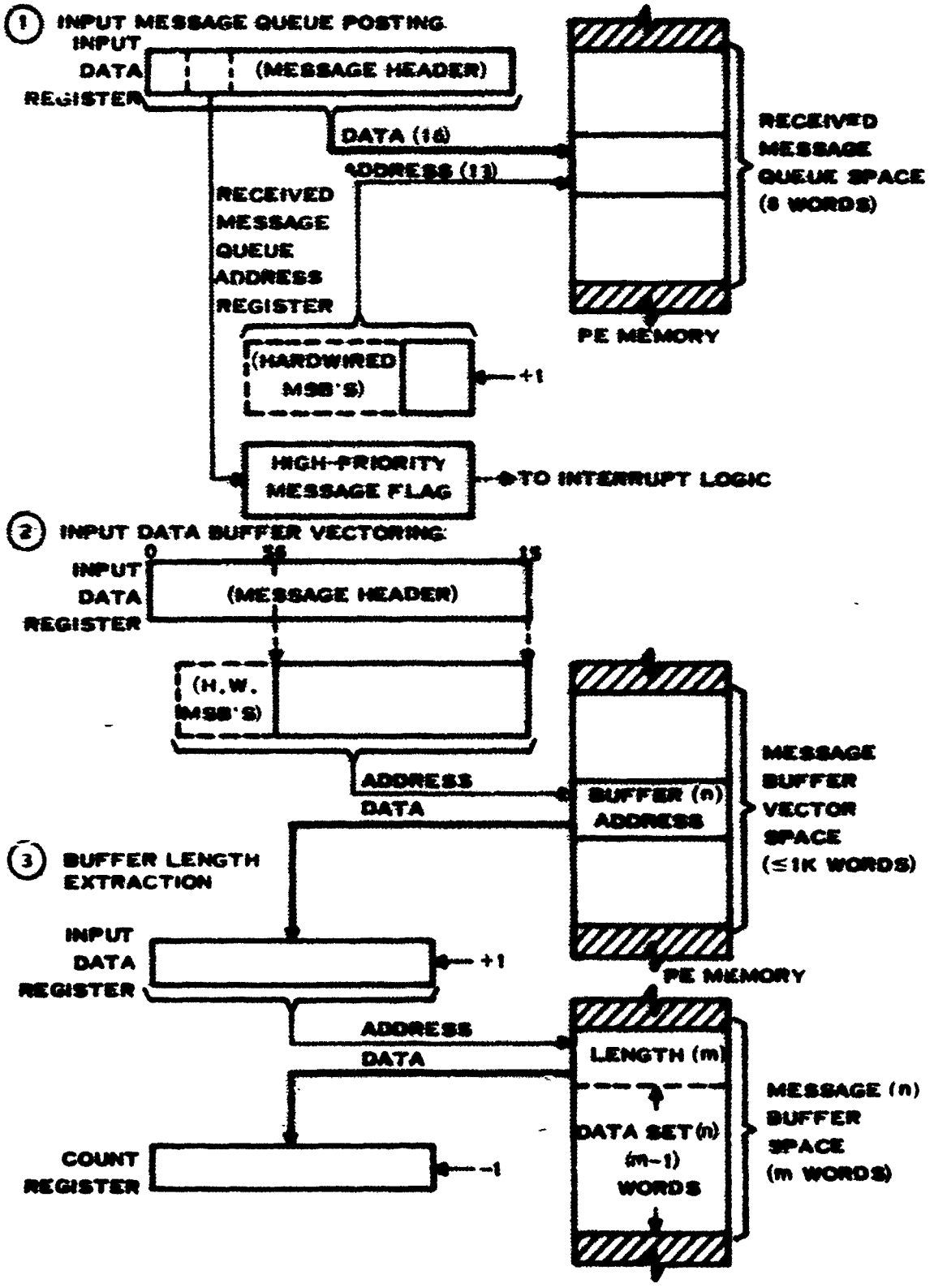



Figure 15. Message Reception Control Initialization Procedure

(reserved) contiguous portion of PE memory referred to as the Message Buffer Vector space. The contents of this location are fetched and used to directly specify the predetermined starting location of the message's data buffer area in memory. Note that the Message Buffer Vector space may be a maximum of 1,024 words in length (corresponding to a maximum permissible 1,024 messages in the system). However, any given system application will require "dedication" of only a portion of this memory space within a given PE equal to the number of messages to be input by the particular PE. This results in "scattered" dedicated or pre-defined locations within this possible 1K word space. Use of non-dedicated space within this area is at the option of the system software linking-loader operation. It is important to note also that access validity into this address space by the Bus Input Channel Controller is guaranteed by the preceding message input selection activity which effectively ensures the validity of the header address if a response match is found (i.e., the M.I.D. associative matching technique is actually a fault-tolerance mechanism associated with this Bus Interface Unit memory addressing function).

The contents of the Message Buffer Vector word fetched in the above step are now used as a PE memory address which corresponds to the first location in the message data buffer area. The content of this initial word, by convention, specifies the correct message length and is fetched by the Bus Input Channel Control logic and loaded into the Input Length Register (ILR).

The input channel is now ready to transfer incoming data words, as they arrive serially, and place them contiguously into PE memory, incrementing the Input Address Register (IAR) and decrementing the Count Register as each word is transferred. Transmission is terminated with the occurrence of either a message sync signal detection on the bus, and/or a Length Register content of "zero." The first of the conditions to occur will cause termination at that point; if the two events are not "simultaneous," an error condition is flagged via an interrupt to the processor. If the termination events are "simultaneous," an appropriate "input message received" interrupt is generated. Message termination also results in the incrementing of the Input Queue Pointer (IQP).

Two interrupts can be generated by the successful completion of the input data transfer. A normal "input message received" interrupt is generated subsequent to every message completion. A "high-priority message received" interrupt is generated after the completion of a message which contains a "true" high-priority tag bit contained in the message header.

A timing analysis of the above described input message transfer sequence reveals that single-word buffering in the BIU input data section is sufficient to maintain data validity between receipt of the message header and final usage of its information content in "setting-up" the input channel for data transfer. Maximum time which may be experienced for this sequence is nine memory cycles for the local bus in a worst case memory access timing situation (simultaneous global and local input activity). The next data word is loaded into the buffer 17 microseconds later, a much longer time than represented by nine memory cycles ($\leq$9 microseconds) under presently envisioned bus and memory bandwidth performance figures.

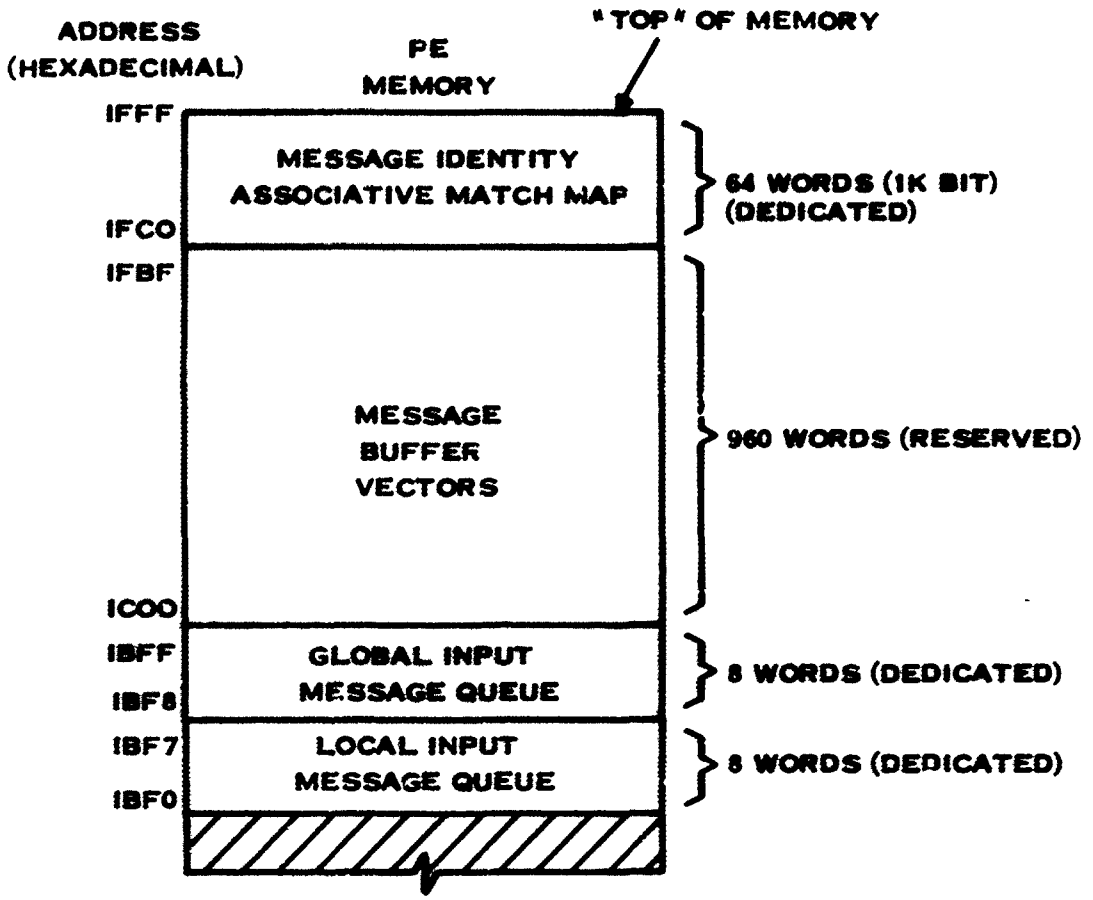



**Figure 16. BIU Memory Allocation Map**

The above message input activities require the use of reserved or dedicated areas of PE memory as described above. Tentative assignments of these areas in memory and, thus, definition of the address generation required of the associated message input control hardware are shown in the memory map diagram of Figure 16.

The above described functional capabilities implemented in the Message Selection Control and Bus Input Channel Control hardware sections offer a quick-reaction, high speed, efficient (in terms of conserving processor throughput) input message handling capability within the PE, which allows it to adapt to and perform proficiently in the envisioned asynchronous, loosely-coupled DP/M bus communications environment.

### 3. Message Transmission Control

The Message Transmission Control section performs the tasks of retrieving output message data from PE memory and transferring them to the Biphase Data Encode/Transmit subfunctional element. The operation of this channel is analogous to that of a conventional autonomous DMA computer I/O channel. The channel is setup and initiated under program command by loading the Output Address Register (OAR) with the address of the contiguous data block buffer to be output. By convention, the first word in this buffer area defines the actual data buffer length. This control data is extracted and loaded into the Output Length Register (OLR). The actual output message data block follows, with the initial word in this block being the message header. (Refer to Figure 17 for a graphical description of this operation.) The channel then oversees the process of receiving each successive word as the previous word is emitted to the bus. Single-word buffering of memory output data is used to maintain a continuous serial data stream output to the bus. At the completion of the buffer transfer, the Bus Output Channel Controller directs the output transmitter to issue a message sync signal and informs the processor of the buffer completion via an interrupt.

Also provided in this functional element is the housing facility fault-tolerance mechanism which detects attempted violation of the eight-word-maximum Global message protocol. Upon detection of an attempted violation (OLR loaded with value greater than eight), the Bus Output Channel Controller aborts the output initialization sequence and generates an interrupt stimulus to the processor to denote the occurrence of the attempted violation.

### 4. Bus Access Control

The Bus Access Control functional element controls the access of the PE data transmission section to the bus. Access control is based on a hardware implemented round-robin circular priority sequence which allows the PE to control the bus when its appropriate allocation slot is detected. Up to 256 bus access slots can be accommodated, thus permitting a satisfactory degree of supercommutation of available bus access slots in envisioned system applications (less than 32 PEs). This control section is program initialized by software commands which define the "length" of the bus (the PE's access control period in units of allocation time slots) and the current "position" of the control slot in the bus control chain. The functional supercommutation of bus allocation time slots among the various users is accommodated by this bus control element as shown by the exemplary bus system in Figure 18. This characteristic is considered important in reducing bus access latency time for "high-priority" bus users.

When the bus control logic recognizes that its time slot has begun, it either enables an output data transfer to begin or effects "passing" of bus control to the next allocation slot by commanding the data transmitter section to emit a Message Sync signal, depending on whether the Output Channel has been activated by the program. This output control logic is also responsible for observing the intermessage gap-timing requirement of 2 microseconds minimum before beginning data transmission.

Also included in the Bus Access Control functional element are the "watchdog-timer" mechanisms provided for bus facility fault detection. These mechanisms are:

Bus Quiescence Watchdog Timer measures time between bus activities (i.e., "no-signal" time). If a maximum allowable quiescent time of 50 microseconds

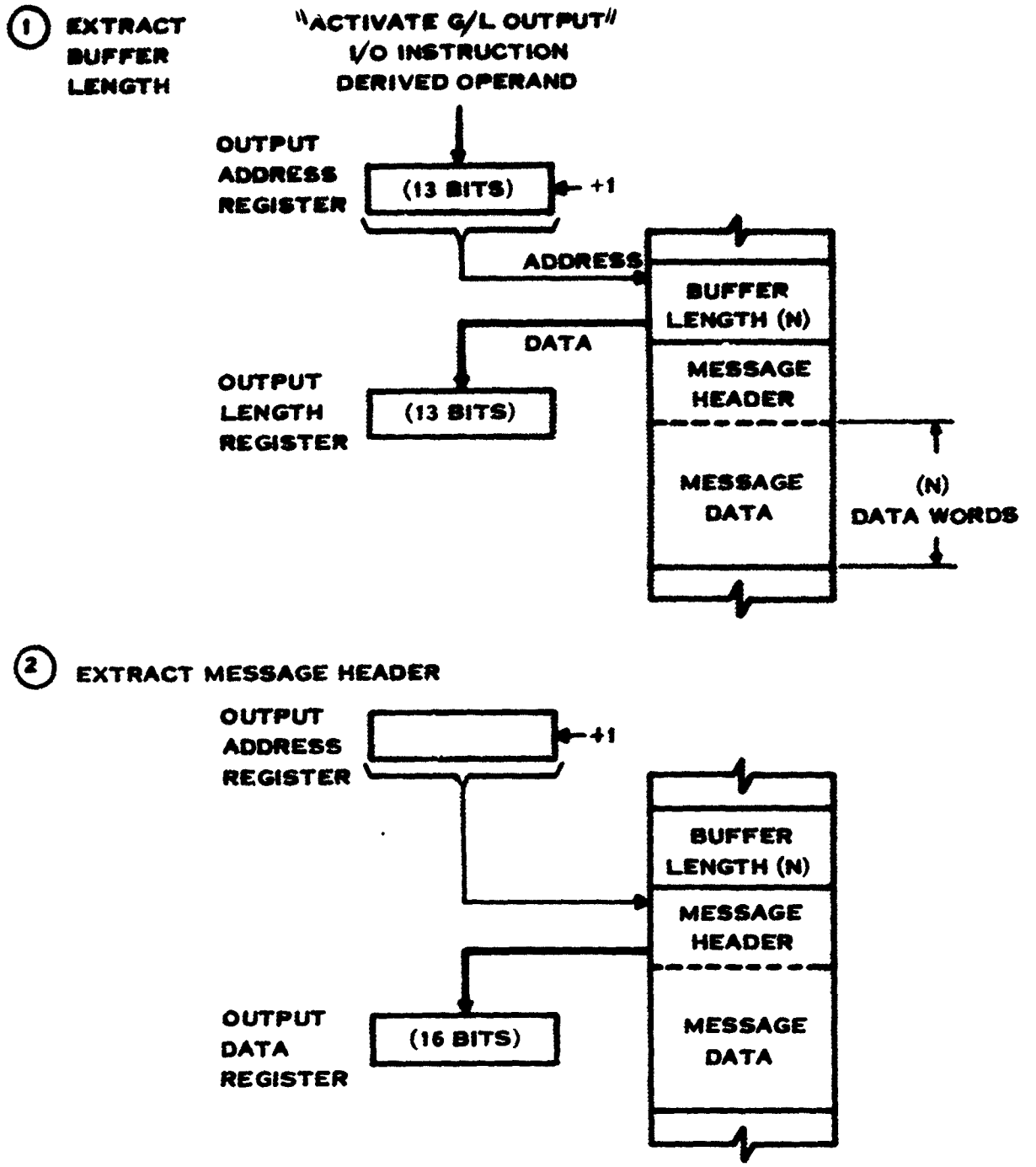



Figure 17. Message Transmission Initialization Procedure

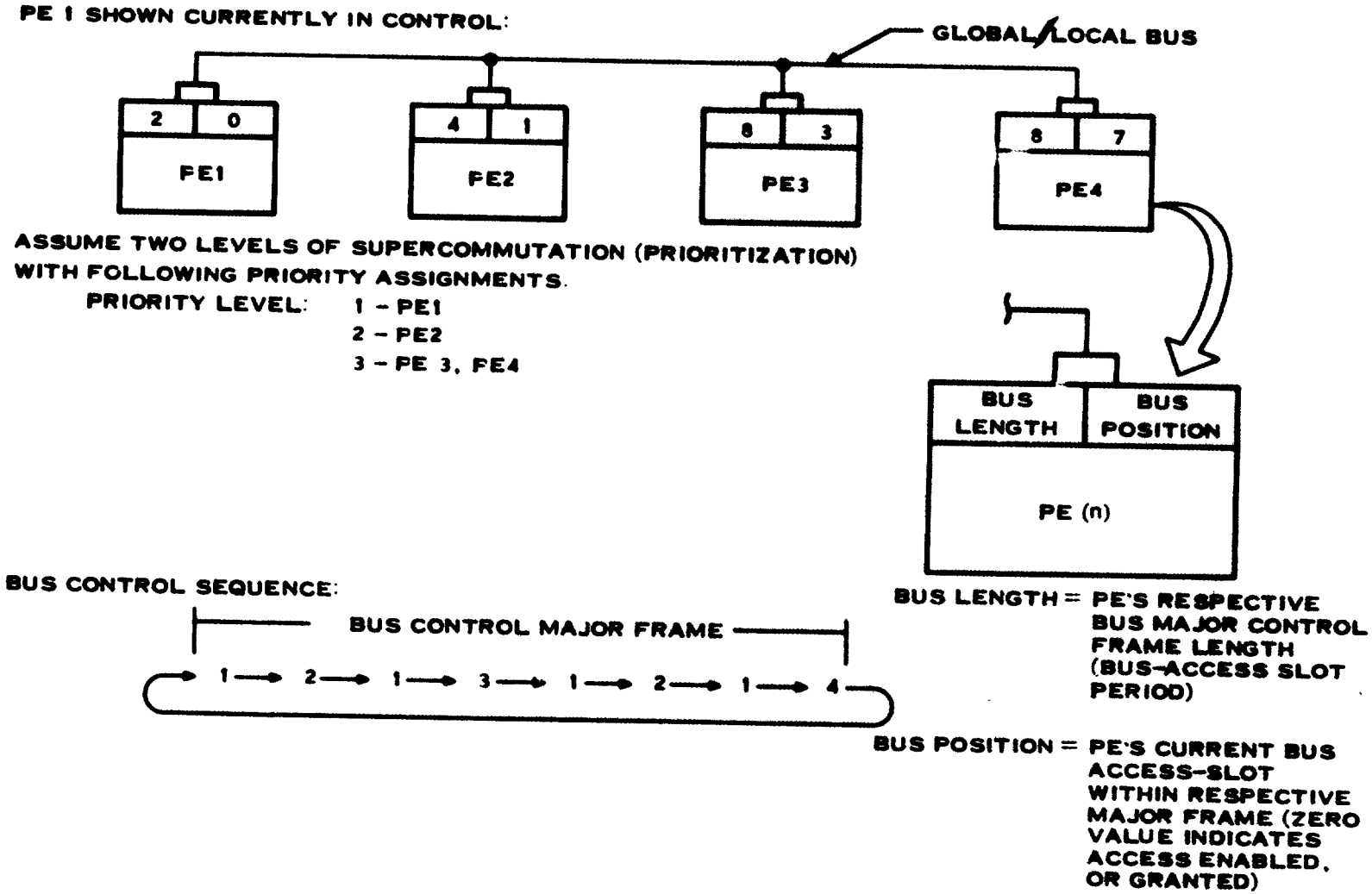



Figure 18. Bus Control Sequence (Supercommutation Example)

is exceeded, an appropriate interrupt stimulus is generated and all Bus Access Control functions are disabled.

Bus Dominance Watchdog Timer measures length of each individual Global message transmission. If a message transmission of greater than 8 data words is detected, an interrupt stimulus is generated and all Bus Access Control functions are disabled.

### 5. Processor/Memory Interface Control

This functional element of the BIU performs the necessary control operations associated with providing interface protocol compatibility between the BIU and the internal PE data exchange bus (I-BUS) which interconnects each functional unit of the PE. All data/command exchange between the processor/memory and the BIU are accomplished via this internal bus. The P/M Interface Control section contains the logic associated with decoding/recognizing BIU control instruction commands from the processor and effecting their execution within the BIU. A list of BIU command/control instructions is shown in Table 4. This functional element also contains the hardware elements required to retain and present the various BIU interrupt stimuli to the processor interrupt input via the I-BUS. The section performs program-controlled masking of each separate interrupt and provides protocol compliance with the processor interrupt-related operations. BIU interrupt stimuli are listed in Table 5.

### 6. Redundant Bus Management

The Redundant Bus Management (RBM) functional element of the BIU is responsible for providing the necessary control elements required to interface a single BIU (and, thus, single PE) to dual-redundant (pair) Global busses. The control logic of this functional element acts as a passive monitor with regard to the designated alternate, or "back-up," bus. This monitoring activity includes "listening" for a valid predetermined (hardware) "switchover" command occurring on the alternate bus. For purposes of fault-tolerance enhancement, the reliability of this detection process may be increased with the use of various validity-checking codes or "pump-up" schemes applied to the formatting and transmission of the "switchover" command message. The required validation technique will then be implemented in the RBM functional element hardware. When a valid "switchover" command message has been recognized, the RBM then switches the BIU Bus Input Channel over to the alternate bus so that all future message receptions are extracted from the newly established bus connection. This action forces the Bus Input Channel logic to abort any input message activity in progress at the time of "switchover" command detection and "resets" all message input control logic functions.

In addition to the above monitoring process, the RBM functional element of the BIU provides the capability to switch the BIU Bus Output Channel to either of the Global busses under program (processor) control. This switching activity can be performed between bus connections as many times as directed by program command. Thus "primary" and "alternate" Global bus identities can be established dynamically and interchanged as many times as required during system operation.

## E. BUS INTERFACE UNIT IMPLEMENTATION CONSIDERATIONS

Implementation requirements of the previously established and discussed functional characteristics of the BIU were next investigated to assess the impact of these characteristic

**TABLE 4. BIU COMMANDS (INSTRUCTIONS)**

**Processor Output (Write):**

1. Load Global Bus Control Parameters (Bus Length and Position)
2. Load Local Bus Control Parameters (Bus Length and Position)
3. Activate Global Output
4. Activate Local Output
5. Load Global Interface Status
6. Load Local Interface Status
7. Load Global Primary Interrupt Masks
8. Load Local Primary Interrupt Masks
9. Enable Global Output Activity
10. Enable Local Output Activity
11. Disable Global Output Activity
12. Disable Local Output Activity
13. Switch Global Output to Alternate Bus

**Processor Input (Read):**

1. Send Present Global Bus Position
2. Send Present Local Bus Position
3. Send Global Input Queue Pointer
4. Send Local Input Queue Pointer
5. Send Global High-Priority Interrupts
6. Send Local High-Priority Interrupts
7. Send Global Low-Priority Interrupts
8. Send Local Low-Priority Interrupts
9. Send Global Interface Status
10. Send Local Interface Status
11. Send Global Primary Interrupt Masks
12. Send Local Primary Interrupt Masks

Note:
Global/Local Status Information contains:

- Bus Interface Enabled/Disabled condition
- Output Channel Busy/Available
- Interrupt Mask register contents

**TABLE 5. BIU INTERRUPTS**

**Global High-Priority:**

1. Global high-priority message received
2. Global bus quiescent
3. Global bus dominated
4. Global message length violation
5. Global message header error
6. Global message word count error
7. Global message word length error
8. Global message parity error
9. Global message encoding error

**Global Low-Priority:**

1. Global message received
2. Global transmission completed

**Local High-Priority:**

1. Local high-priority message received
2. Local bus quiescent
3. Local bus dominated
4. Local message length violation
5. Local message header error
6. Local message word count error
7. Local message word length error
8. Local message parity error
9. Local message encoding error

**Local Low-Priority:**

1. Local message received
2. Local transmission completed

Note: Interrupts are listed in order of decreasing priority

requirements on actual hardware design. Of primary importance in this design investigation was the effect of each BIU functional element on LSI device complexity. One of the underlying criteria of the DP/M Processing Element design was the compatibility of the functional design with LSI device technology

capabilities in both the near future (mid-to-late 1970's) and early 1980 time-frames. Thus, design simplicity at the functional level was sought and favored in many of the tradeoff areas discussed previously.

### 1. BIU Register Level Description

To allow determination of the practicality and extent of LSI implementation achievable with the functional design given, a preliminary register level design of the BIU was performed. This level of hardware design describes each basic hardware functional element required to implement each BIU functional element and thus allows approximate, but representative, quantitative estimates of LSI device complexity requirements. Also defined are the interface signal requirements of each functional element to allow representative estimates of device I/O signal (pinout) requirements.

The register level design of each Bus Interface Unit functional element is shown in the register level block diagrams of Figures 19 through 27. Note that the message selection control and bus input channel control functional subelements have been shown collectively in the message reception control register level description because of their directly related and somewhat common hardware functional requirements. Also, the major hardware sequence control functions required in the Message Reception, Message Transmission, and Bus Access Control functions are represented only at the block level. For complexity estimating purposes, a state flow (flow chart) description which describes the procedure, or state sequence, required of the control logic elements is given for each of these control functions in Figures 21, 23, and 25, respectively.

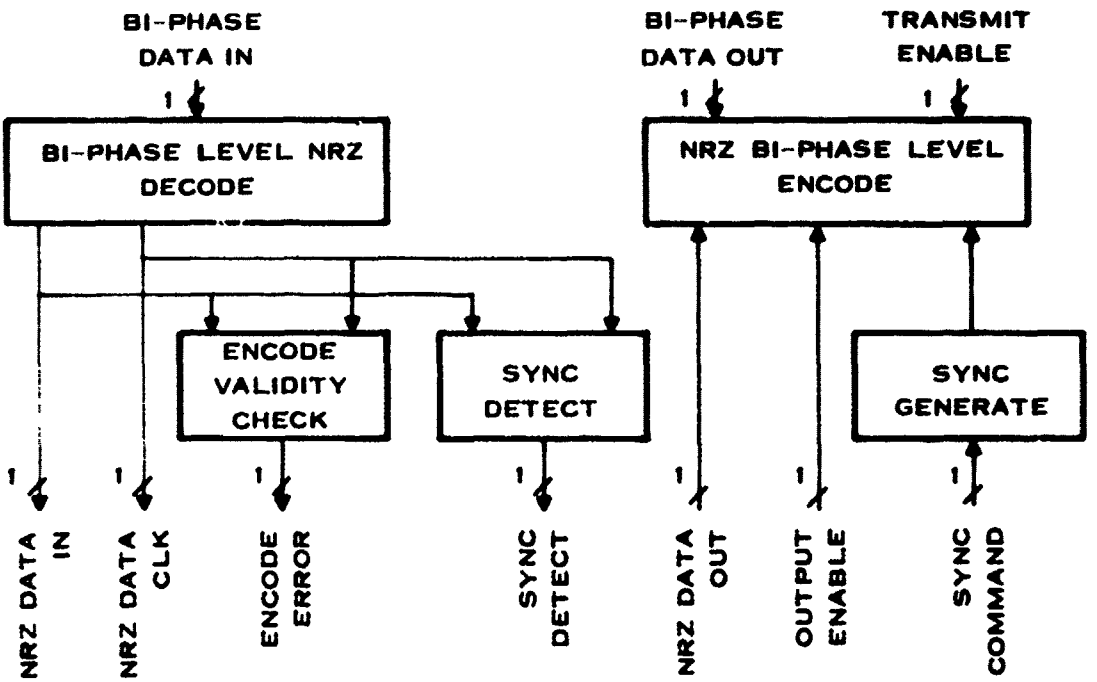



Figure 19. Bus Interface Translation Register-Level Block Diagram

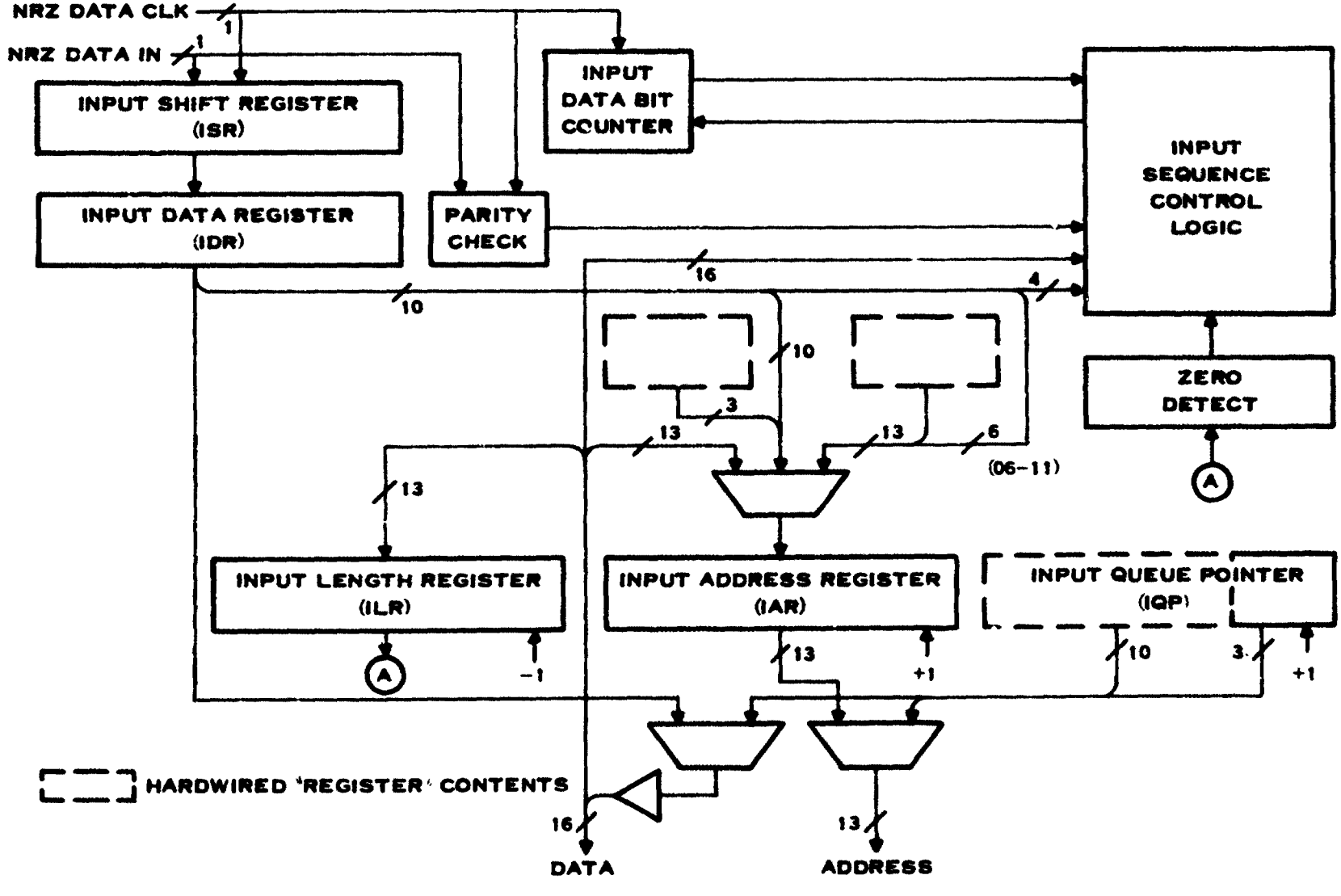
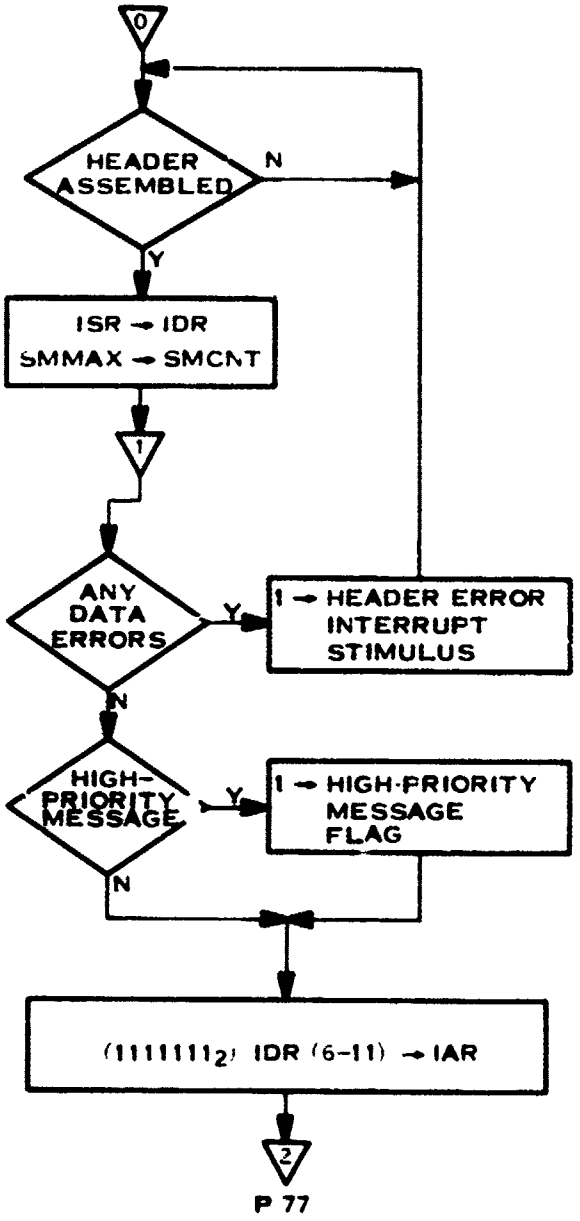



Figure 20. Message Reception Control Register-Level Block Diagram




IDLE UNTIL A MESSAGE HEADER IS RECEIVED

LOAD ASSEMBLED HEADER INTO BUFFER;INITIALIZE MESSAGE SYNC WAIT TIME COUNTER

IF ANY ERROR EXISTS IN HEADER, GENERATE INTERRUPT

IF HEADER INDICATES H.P. MESSAGE, SET H.P. FLAG

GENERATE MIAMM MEMORY ADDRESS (13 BITS) FROM HEADER BITS (6–11), STARTING AT LOCATION IFC0 (HEXA-DECIMAL)

Figure 21 Message Reception Controller Flow Diagram (Sheet 1 of 6)

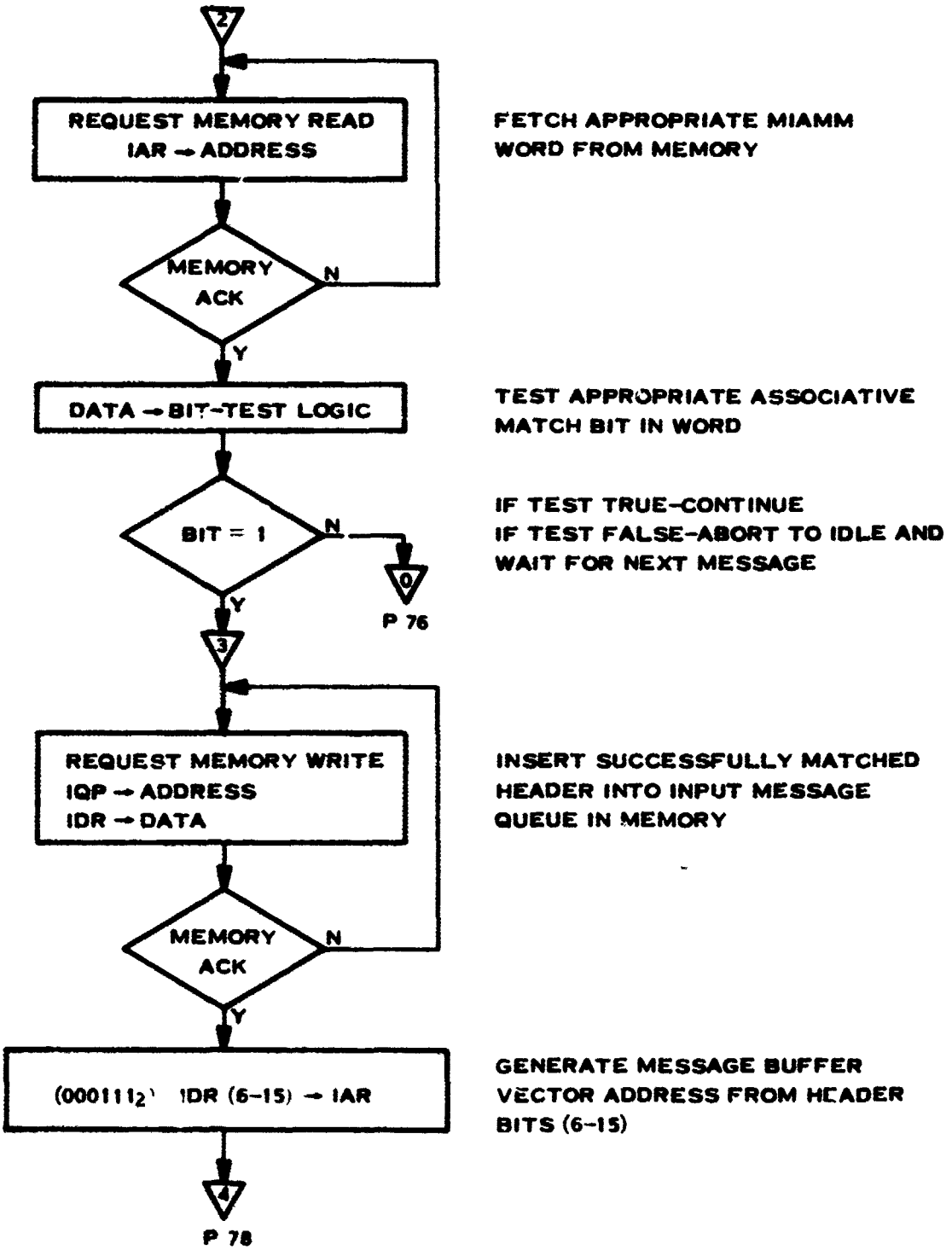



Figure 21. Message Reception Controller Flow Diagram (Sheet 2 of 6)

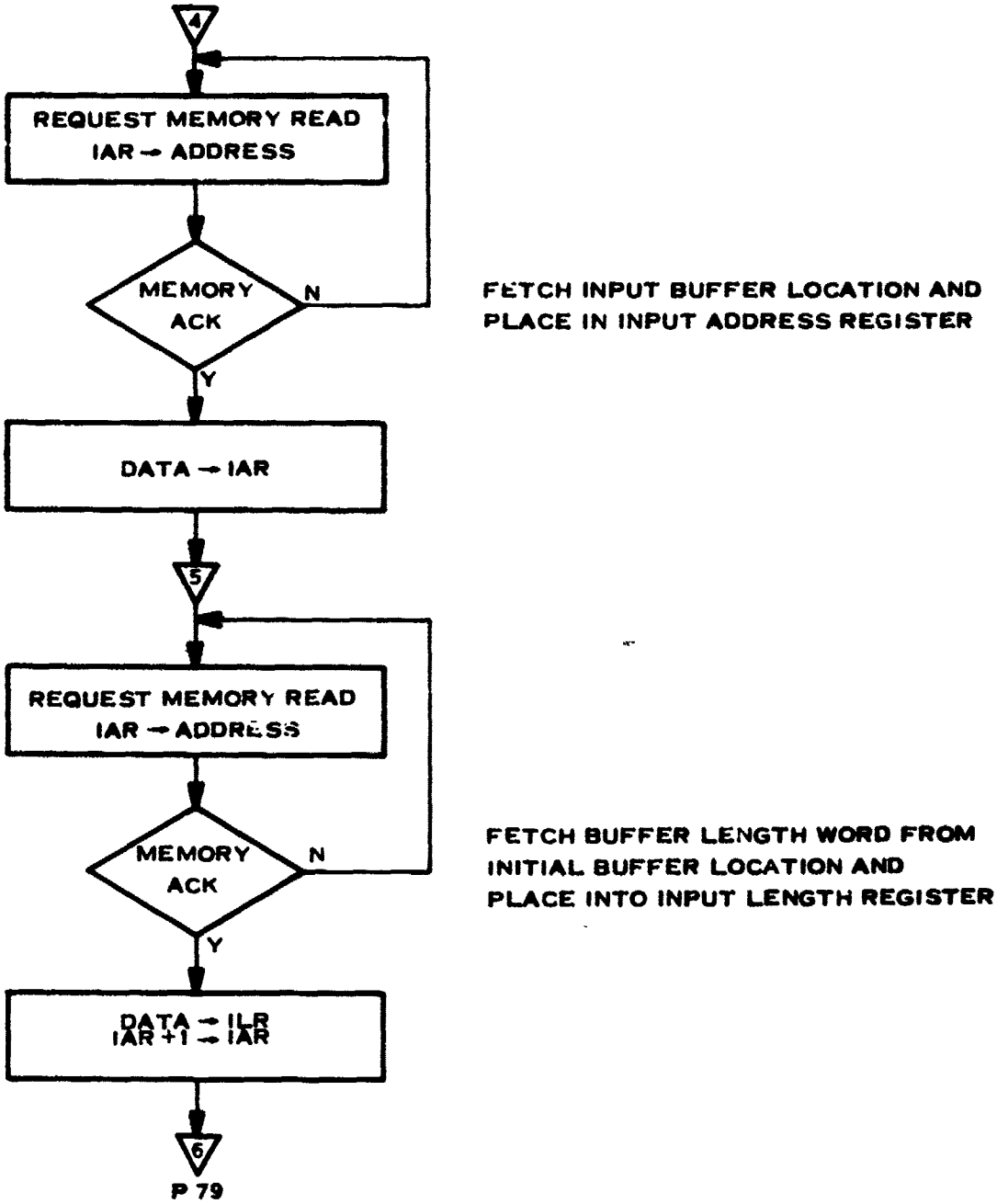
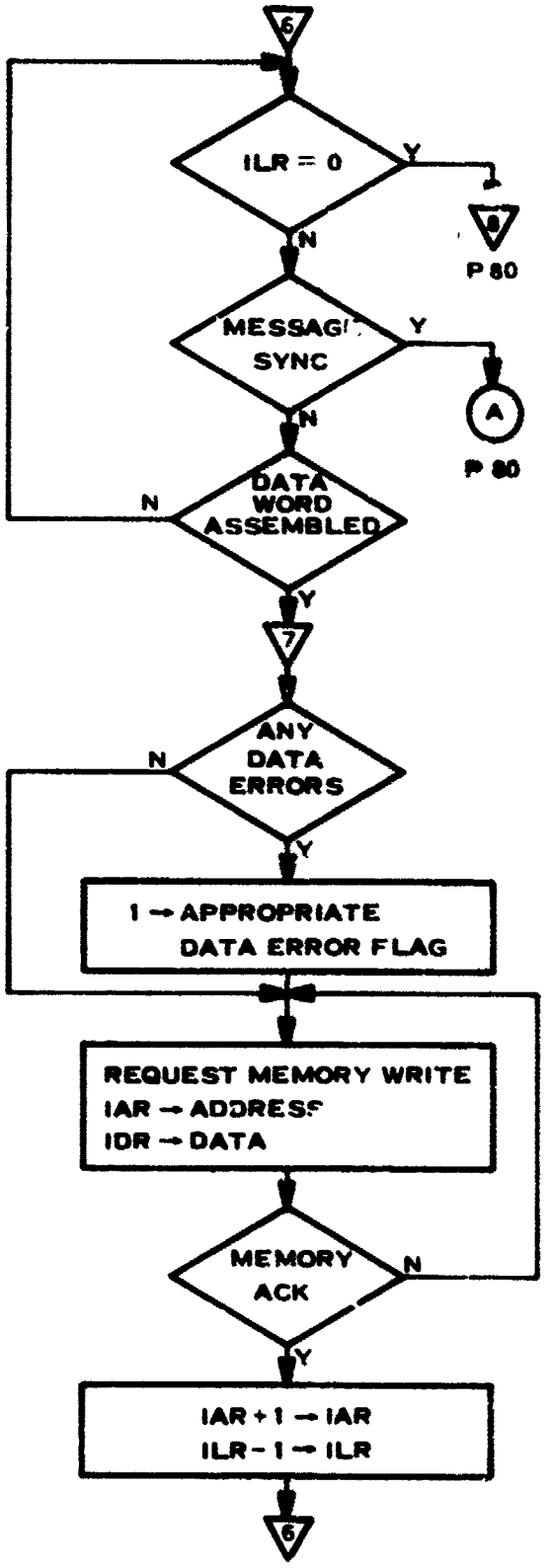



Figure 21. Message Reception Controller Flow Diagram (Sheet 3 of 6)




MESSAGE INPUT TERMINATED BY EITHER ILR DECREMENTED TO ZERO OR MESSAGE SYNC OCCURRENCE

IF DATA ERROR EXISTS IN DATA WORD, SET APPROPRIATE FLAG

WRITE DATA WORD INTO CURRENT MEMORY BUFFER LOCATION

INCREMENT BUFFER LOCATION
DECREMENT WORD COUNT

Figure 21. Message Reception Controller Flow Diagram (Sheet 4 of 6)

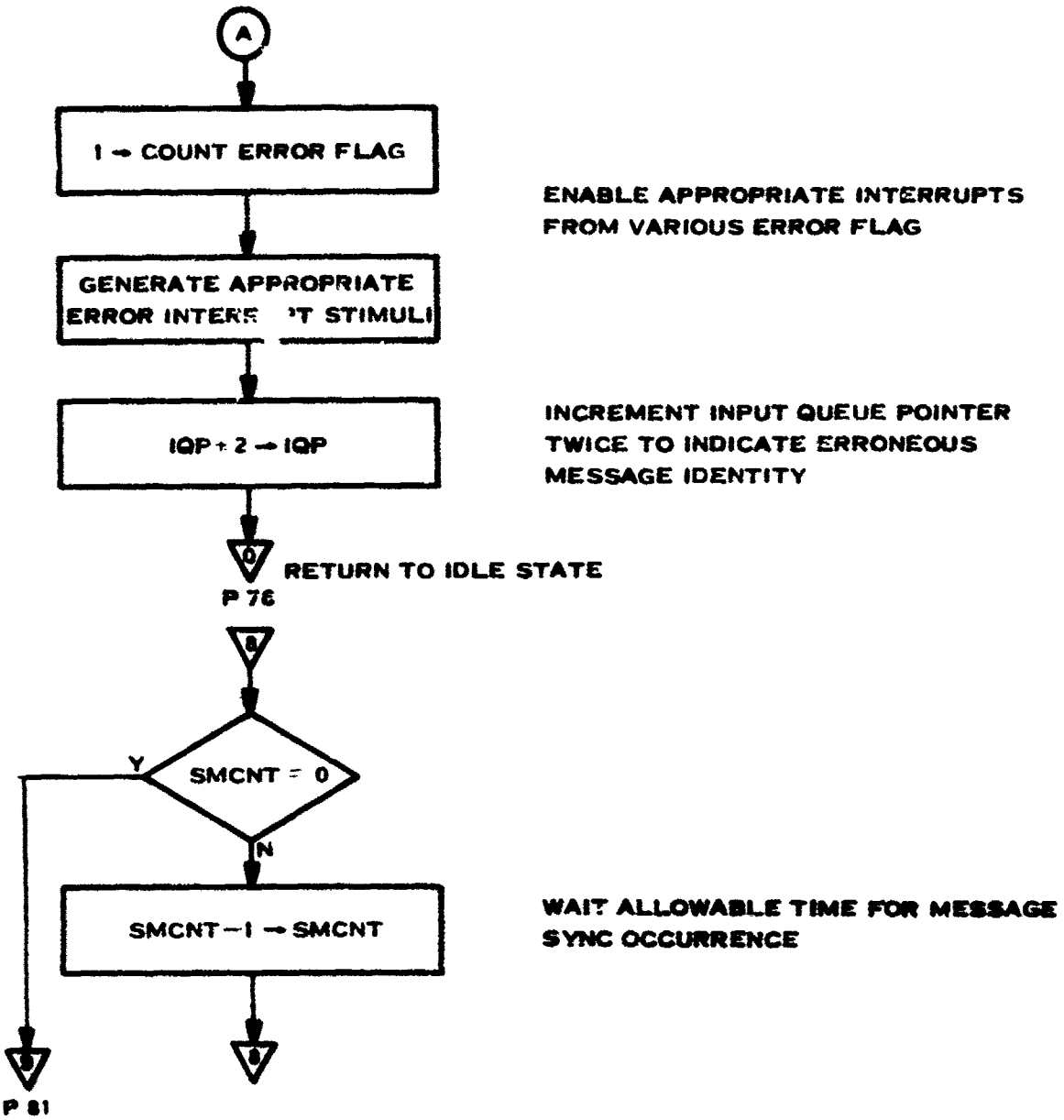



Figure 21. Message Reception Controller Flow Diagram (Sheet 5 of 6)

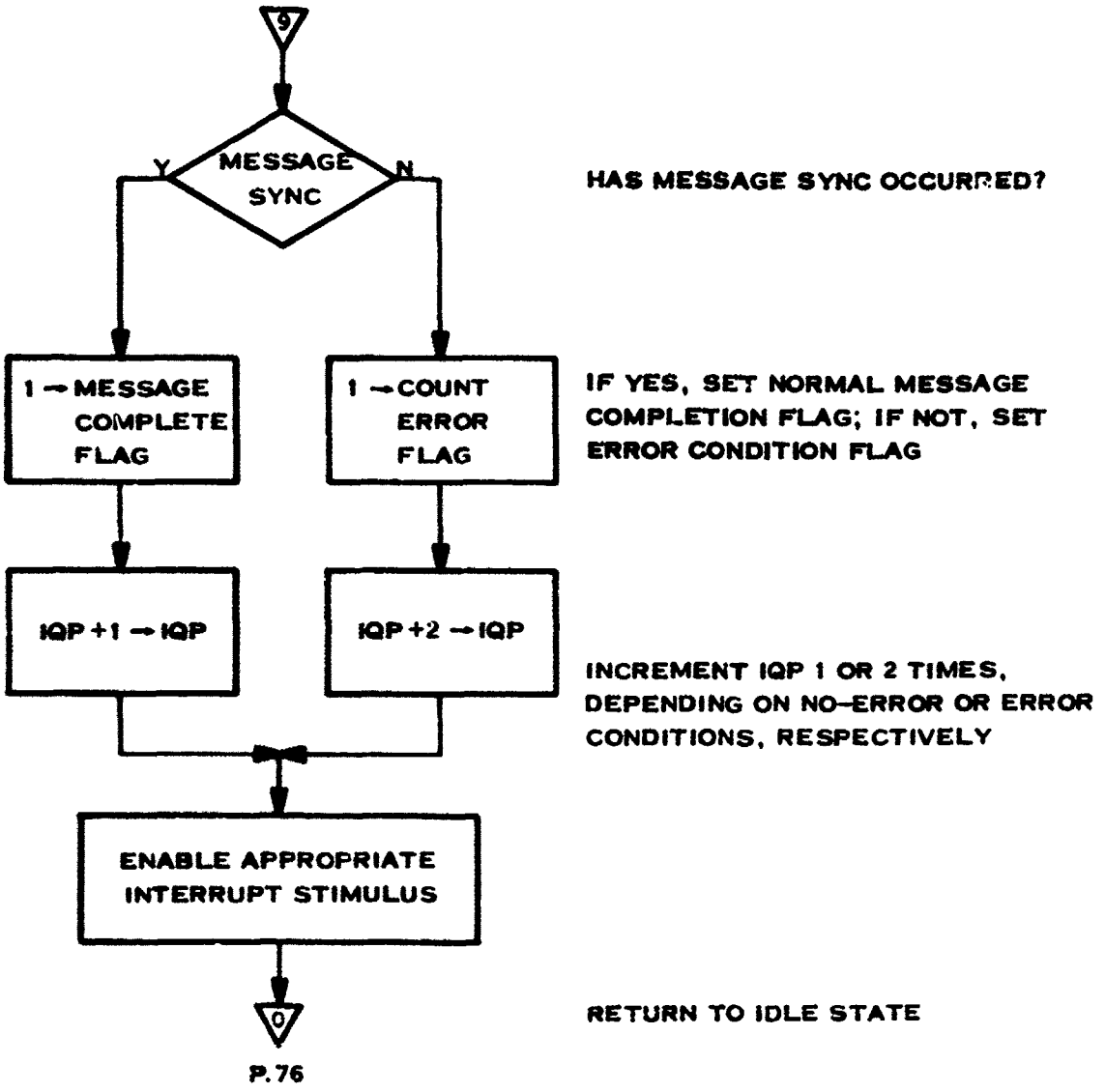



Figure 21. Message Reception Controller Flow Diagram (Sheet 6 of 6)

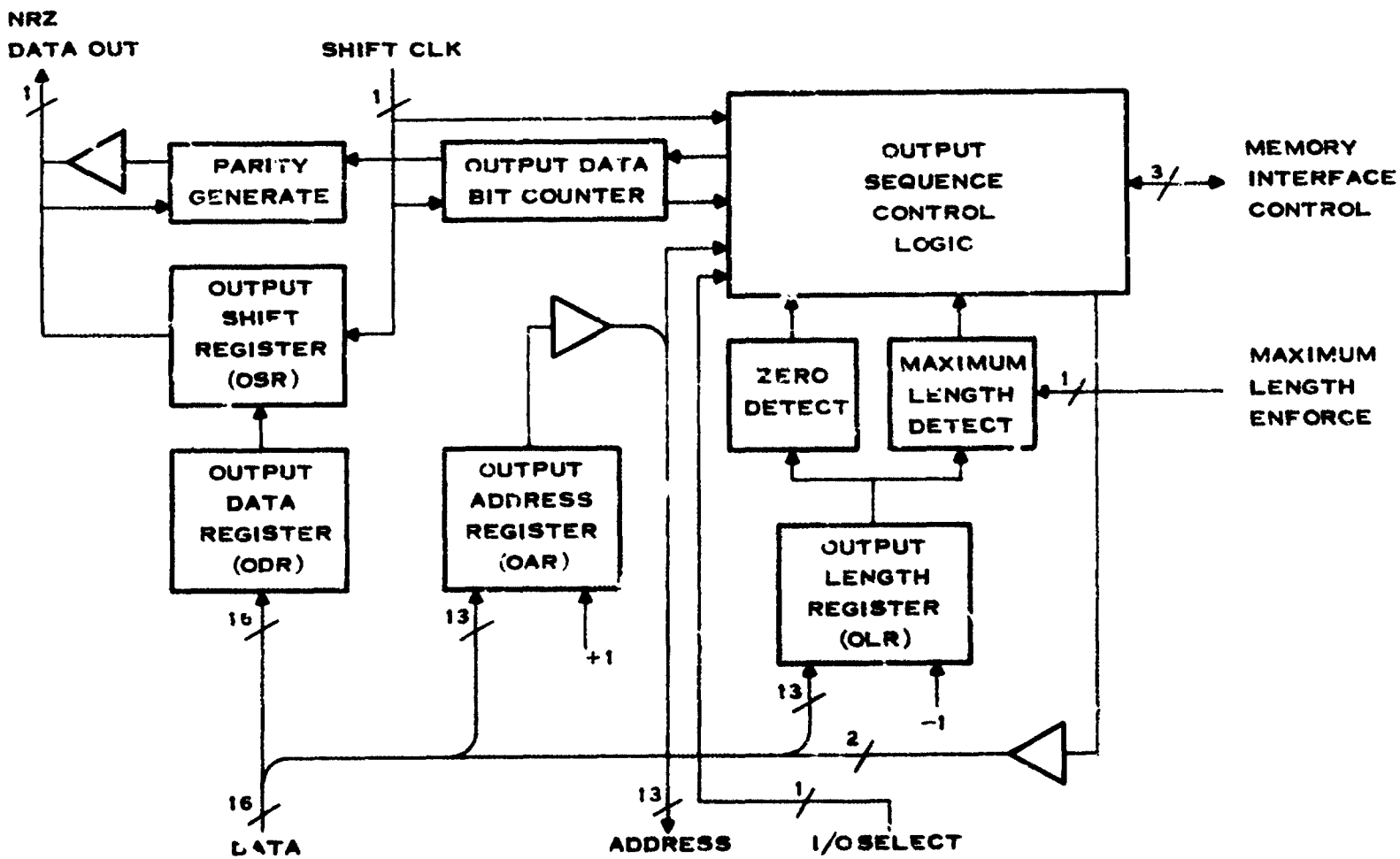
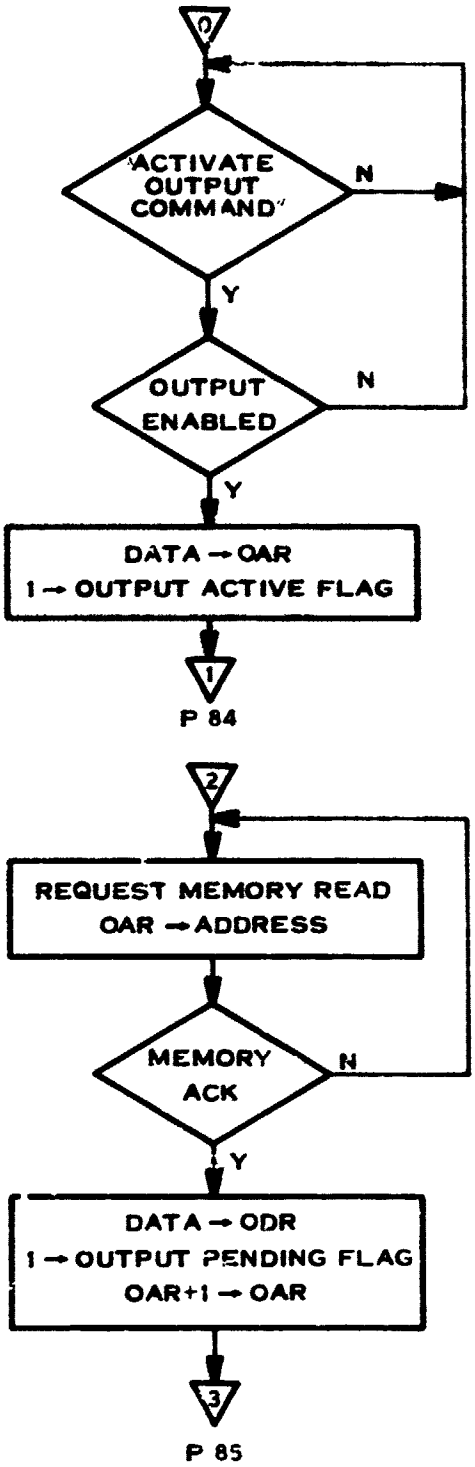



Figure 22. Message Transmission Control Register-Level Block Diagram



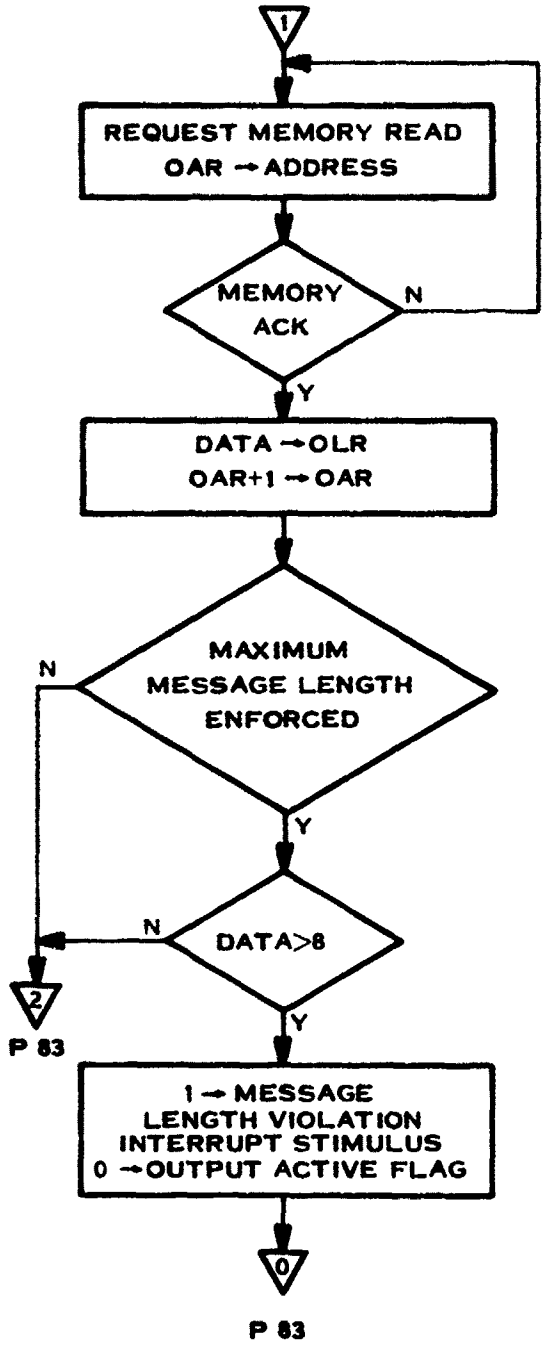

IDLE UNTIL "ACTIVATE OUTPUT" INSTRUCTION IS EXECUTED AND OUTPUT CHANNEL IS ENABLED

LOAD OPERAND (OUTPUT BUFFER ADDRESS) OF "ACTIVATE OUTPUT" INSTRUCTION INTO OUTPUT ADDRESS REGISTER

READ INITIAL MESSAGE WORD (HEADER) FROM OUTPUT BUFFER

PLACE HEADER INTO OUTPUT DATA BUFFER REGISTER: INCREMENT MEMORY ADDRESS

Figure 23. Message Transmission Controller Flow Diagram (Sheet 1 of 4)




FETCH CONTENTS OF INITIAL BUFFER LOCATION (BUFFER LENGTH)

LOAD BUFFER LENGTH DATA INTO OUTPUT LENGTH REGISTER; INCREMENT MEMORY ADDRESS

IF MAXIMUM MESSAGE LENGTH VIOLATION INTERRUPT IS ARMED AND BUFFER LENGTH GREATER THAN 8 IS READ FROM MEMORY, ISSUE INTERRUPT AND ABORT OUTPUT ACTIVITY

Figure 23. Message Transmission Controller Flow Diagram (Sheet 2 of 4)

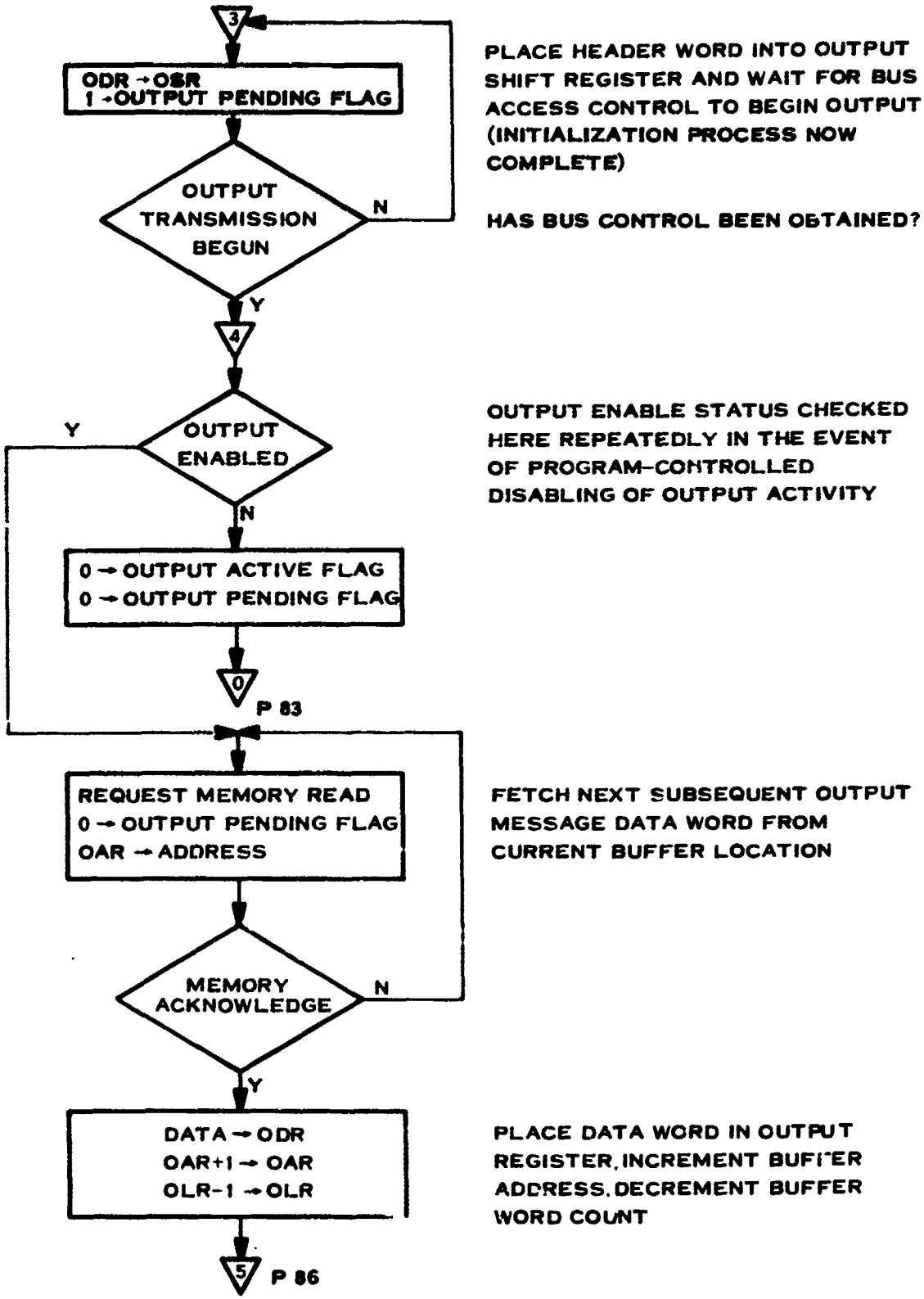
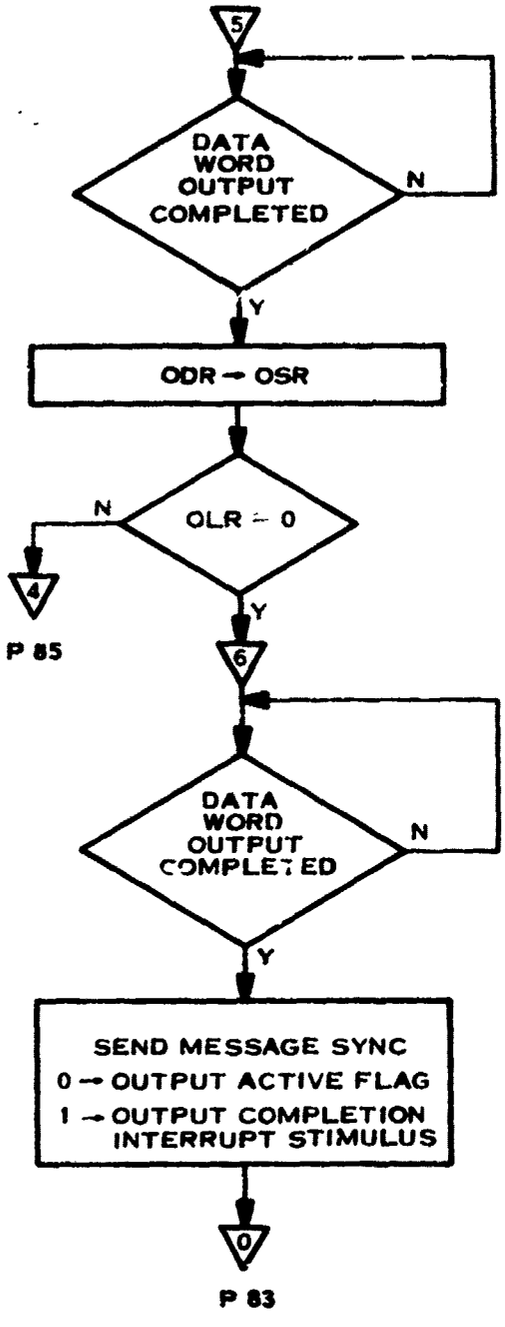



Figure 23. Message Transmission Controller Flow Diagram (Sheet 3 of 4)




AS EACH COMPLETE DATA WORD IS SHIFTED OUT, PLACE NEXT DATA WORD INTO OUTPUT SHIFT REGISTER

TERMINATE DATA FETCH ACTIVITY WHEN WORD COUNT IS DECREMENTED TO ZERO

WHEN FINAL DATA WORD IS SHIFTED OUT, ISSUE MESSAGE SYNC SIGNAL, DEACTIVATE OUTPUT CHANNEL, AND GENERATE COMPLETION INTERRUPT STIMULUS

Figure 23. Message Transmission Controller Flow Diagram (Sheet 4 of 4)

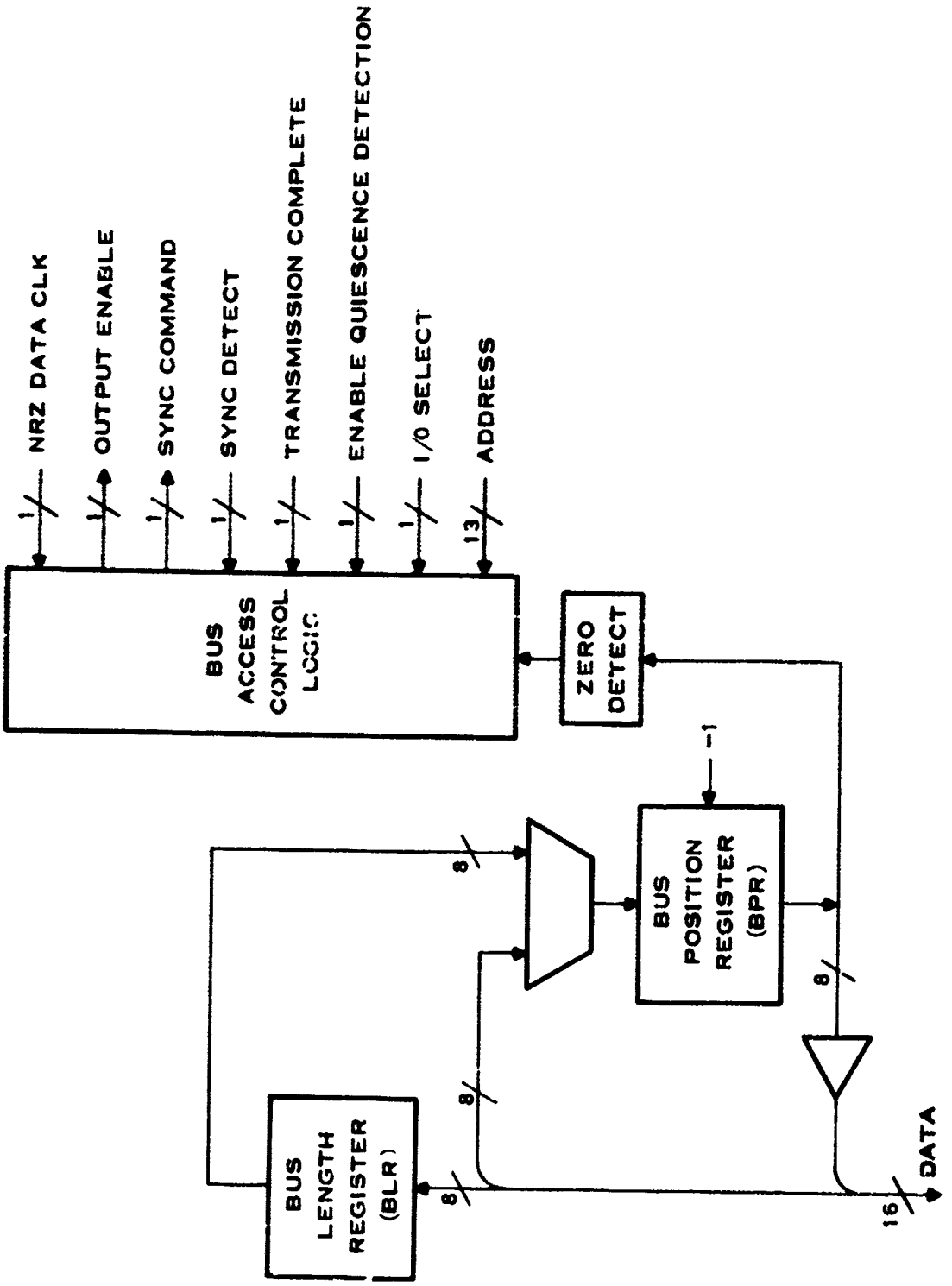



Figure 24. Bus Access Control Register-Level Block Diagram

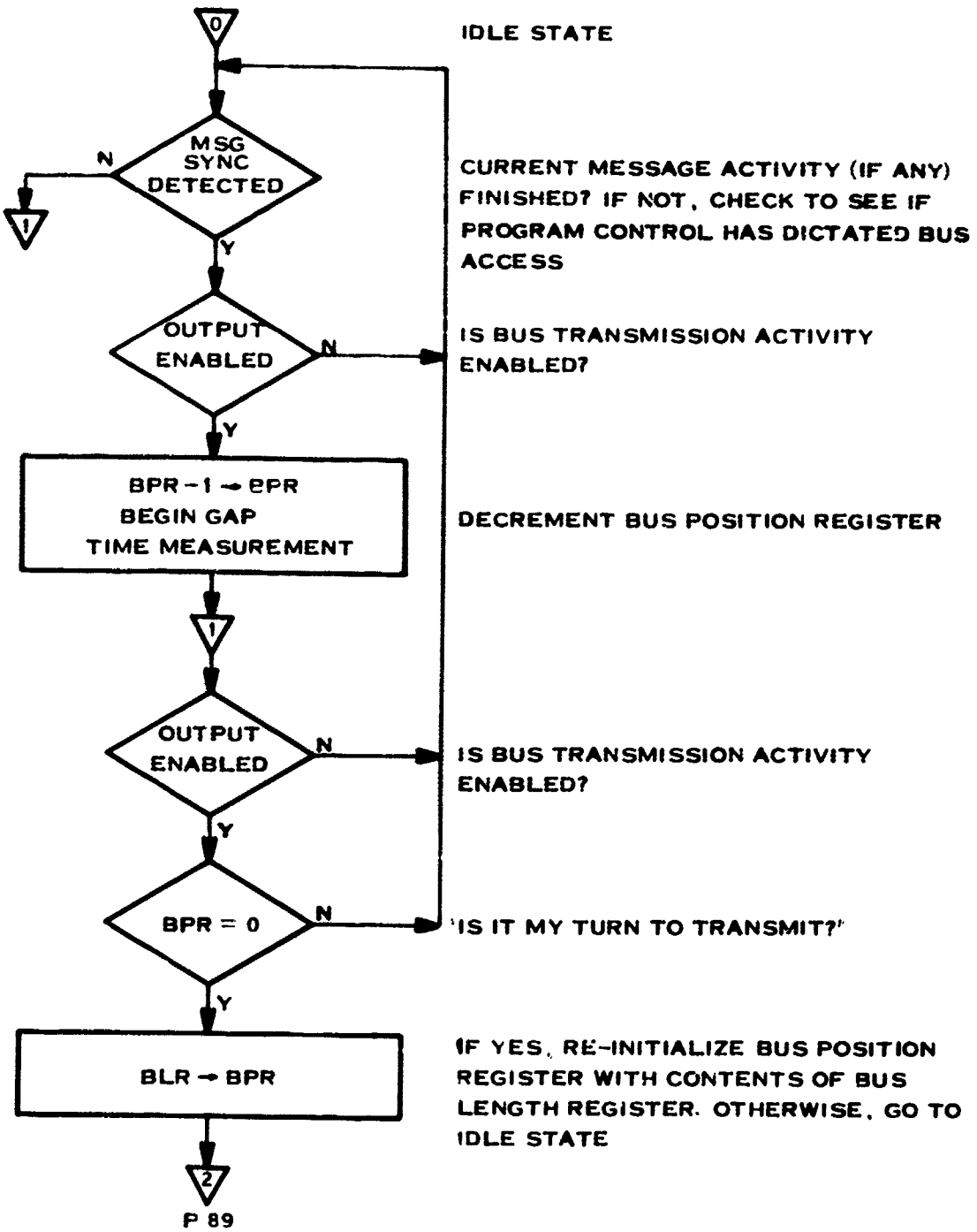



Figure 25. Bus Access Controller Flow Diagram (Sheet 1 of 4)

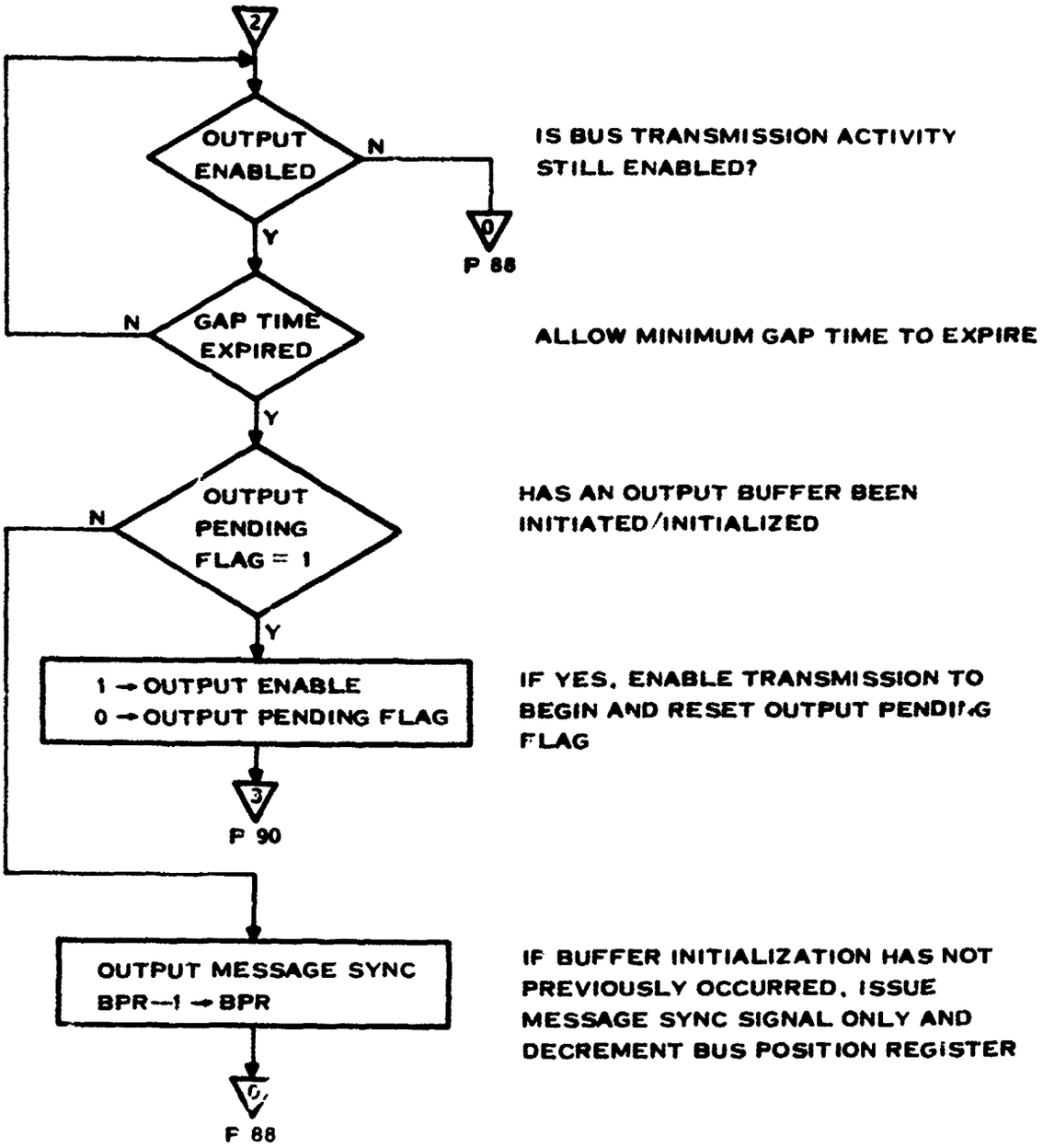



Figure 25. Bus Access Controller Flow Diagram (Sheet 2 of 4)

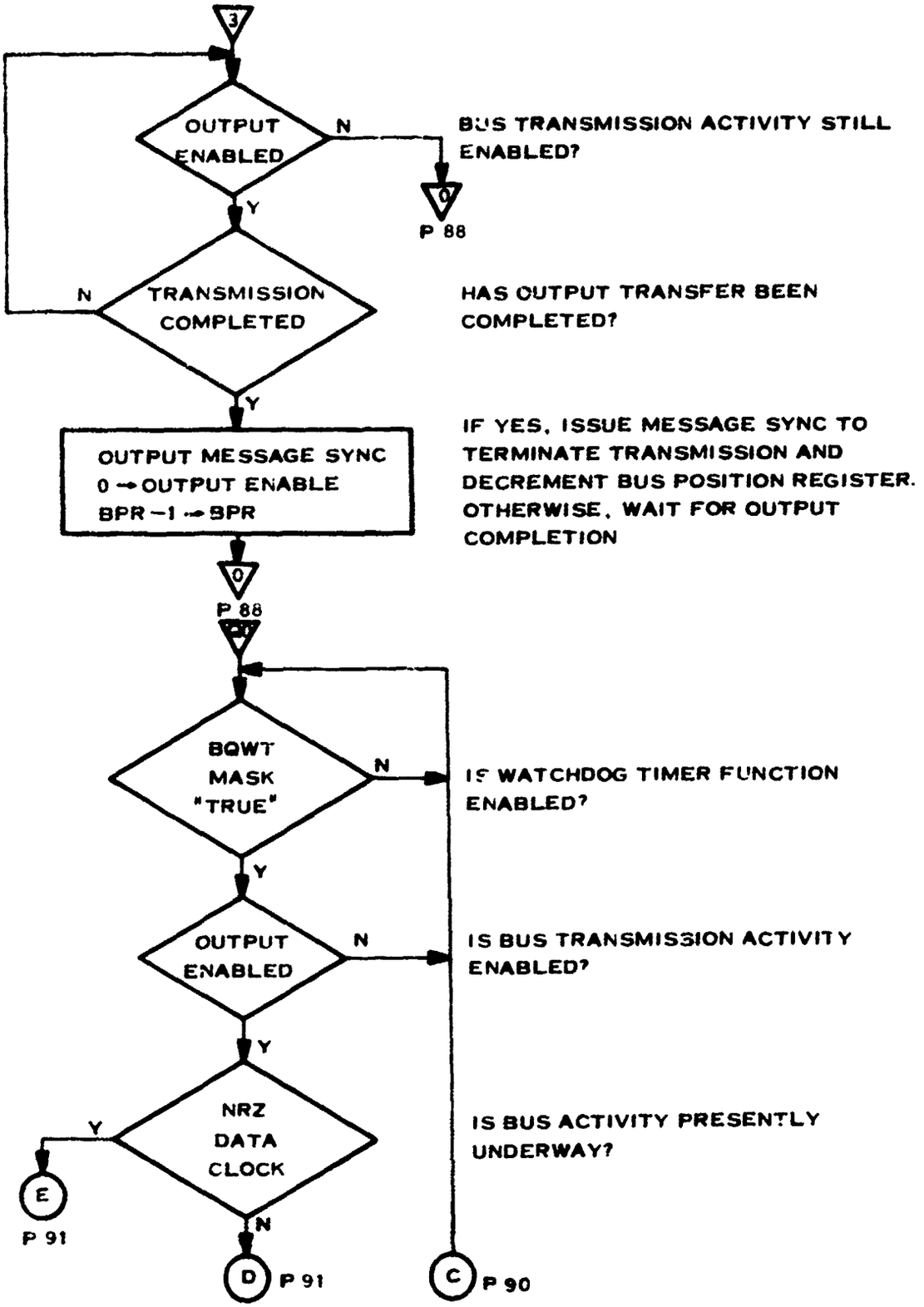



Figure 25. Bus Access Controller Flow Diagram (Sheet 3 of 4)

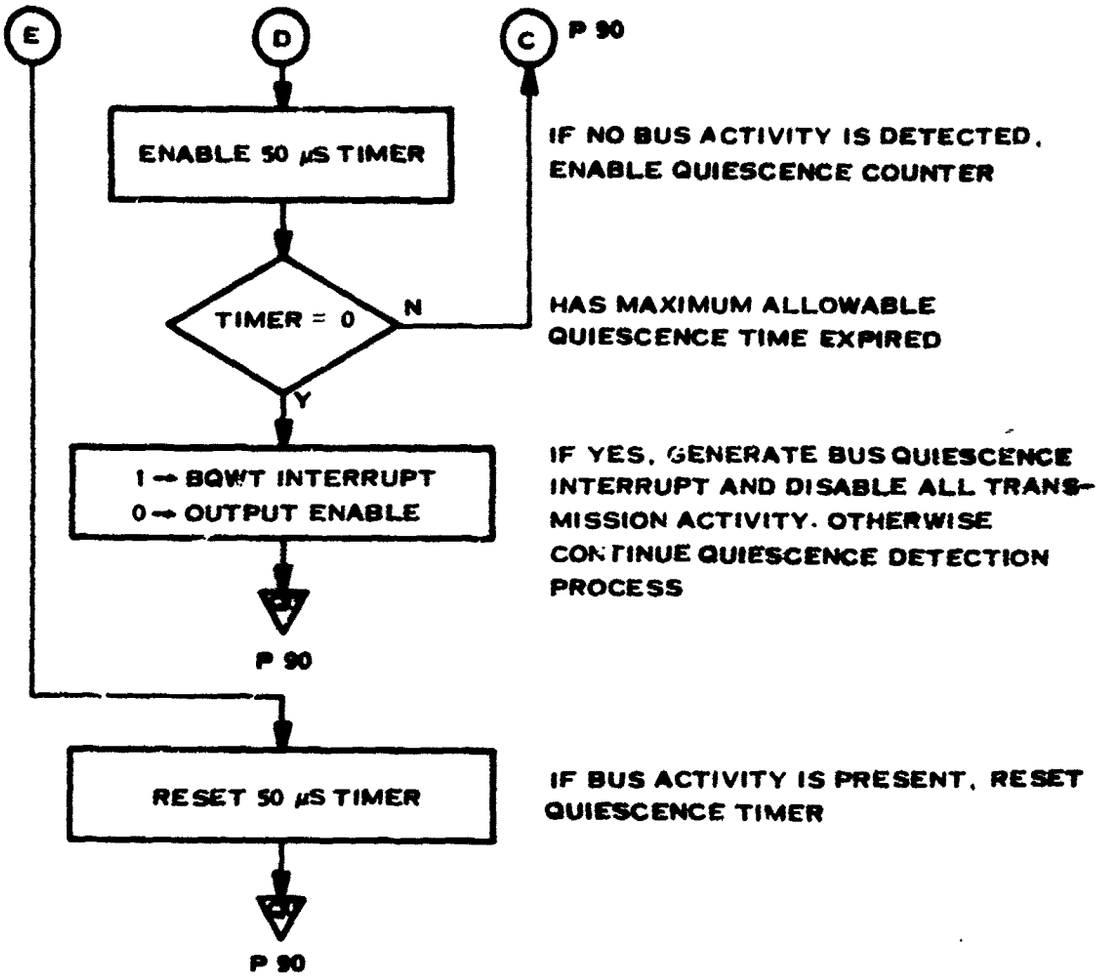



Figure 25. Bus Access Controller Flow Diagram (Sheet 4 of 4)

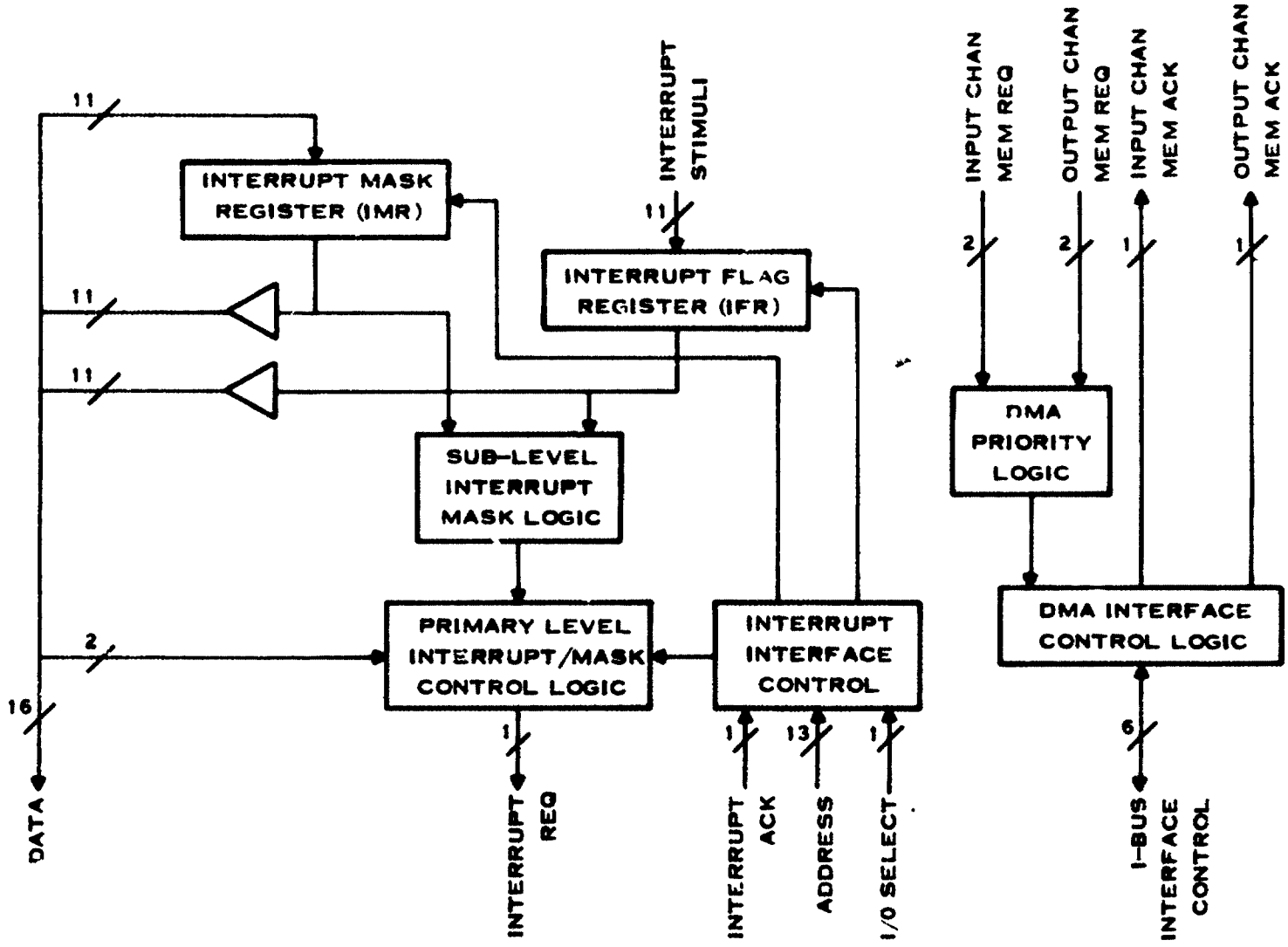



Figure 26. Processor/Memory Interface Control Register-Level Block Diagram

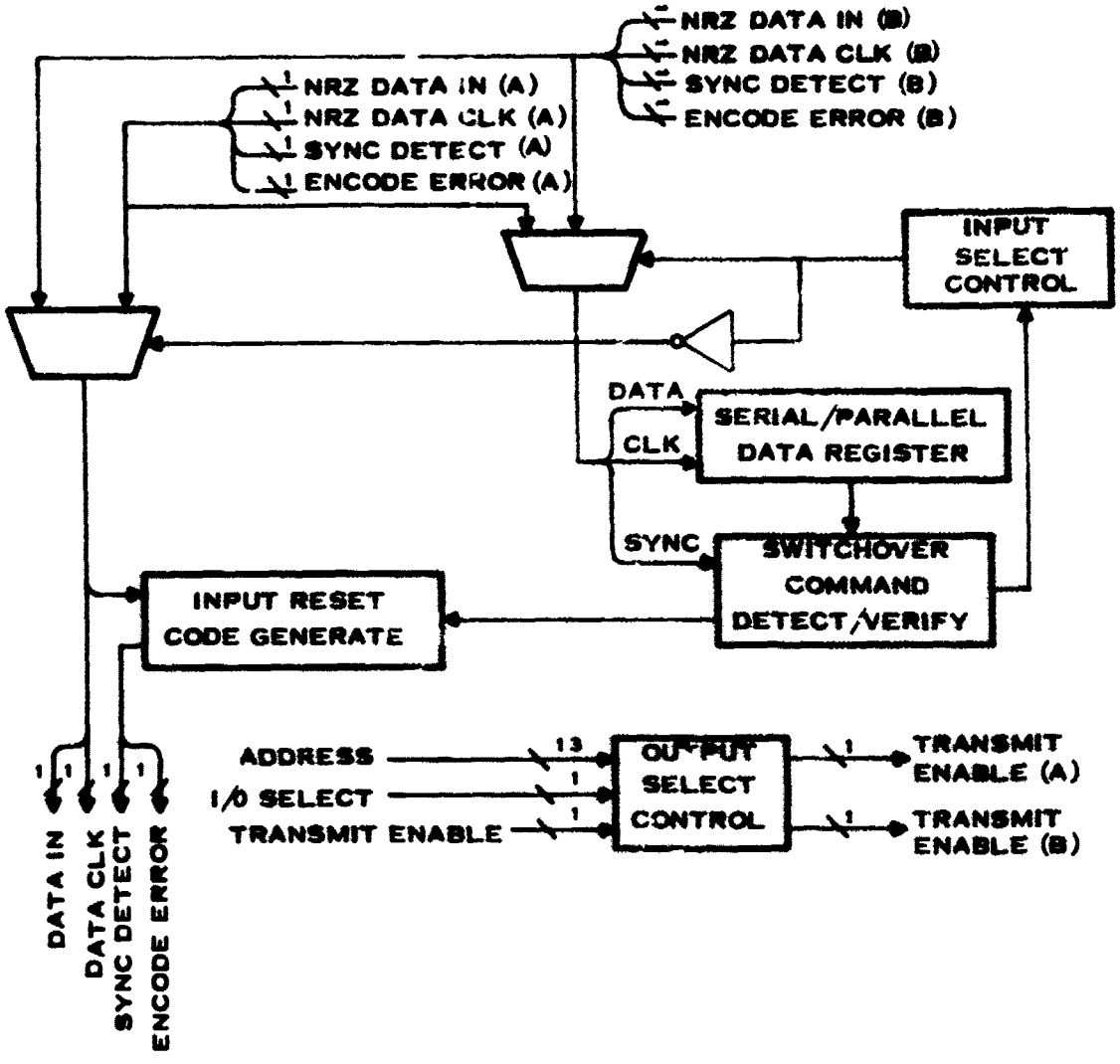



**Figure 27. Redundant Bus Management Register-Level Block Diagram**

It is important to reiterate here that the design of the overall Bus Interface Unit allows the use of a collection of common replicated BIU functional elements to implement both Global and Local bus interfaces. This characteristic is very important and advantageous with respect to detailed hardware implementation considerations as will be discussed in Section VI. Thus, the BIU register level design described herein, with the exception of the Redundant Bus Management functional element, reflects the hardware actually required to interface the PE with each level of bus connection and simply is replicated for both levels (Global and Local) of bus connections. Figures 28 and 29 show the register-level interconnections of these functional elements required to configure the overall Global and Local Bus Interfaces, respectively.

## 2. Hardware Complexity Estimates

The register level design description of the BIU functional elements allows a satisfactory estimate of device implementation complexity requirements to be performed. A logic "gate" was chosen as a standard unit of measure of device complexity; however, the interpretation of an actual "gate" in terms of actual functional and physical attributes is somewhat variable, depending on the semiconductor technology used in the device implementation. Since the characteristics of present generation bipolar TTL technology logic devices are universally known, each BIU functional element was sized in units of TTL-equivalent gates as an interim standard; this choice keeps the sizing estimates from becoming too biased toward any given LSI technology and also affords a unit of measure which may be relatively easily converted quantitatively into appropriate units of measure for the various alternative LSI technologies.

The actual complexity estimate results for the BIU functional elements are given in Table 6. These estimates were formulated from the following sizing quidelines and assumptions:

- Where applicable, catalog-advertised TTL device gate counts were used straightforward to estimate similar BIU functions
- Each hardware register assumes conventional "flip-flop" device implementation with 6 gates required per bit of register storage
- All sequential control functions were assumed to be implemented with conventional encoded-state sequential-controller techniques
- Manchester II/NRZ data code conversion (encode/decode) hardware estimates reflect a conservative best "guesstimate" of worst-case implementation based on examination of possible alternative implementation techniques (e.g., frequency-independent, or frequency-dependent)
- A "blanket" estimate factor of 20 percent was used to estimate the amount of ancillary logic elements (clock-logic drivers, clock gaters, etc.) required in each functional element implementation as a function of the amount of control-only logic utilized in the functional element.

There is one area of potential significant BIU complexity reduction worthy of note. If overall system fault-tolerance considerations allow, the BIU device complexity can be reduced to approximately 1,525 gates, a reduction of 475 gates (or approximately 25 percent of overall BIU complexity) by implementing the BIU as a half-duplex instead of full-duplex data transfer facility. This functional modification would remove the capability to use the BIU in a "self-test" mode since simultaneous message transmission and reception could not be facilitated. Recommendation of this modification cannot be established until the actual impact of the more complex design on LSI implementation considerations are more well-defined.

## F. BIU SIMULATION MODELING

A functional simulation model reflecting the current BIU register level design was developed as a portion of the PE functional simulator. This model accurately represents the operation of the BIU hardware at a functional register level. The BIU model is modularly structured as the integration of a set of BIU-event models, each of which is a FORTRAN subroutine representation of a discrete functional operation. Each event is modeled in terms of

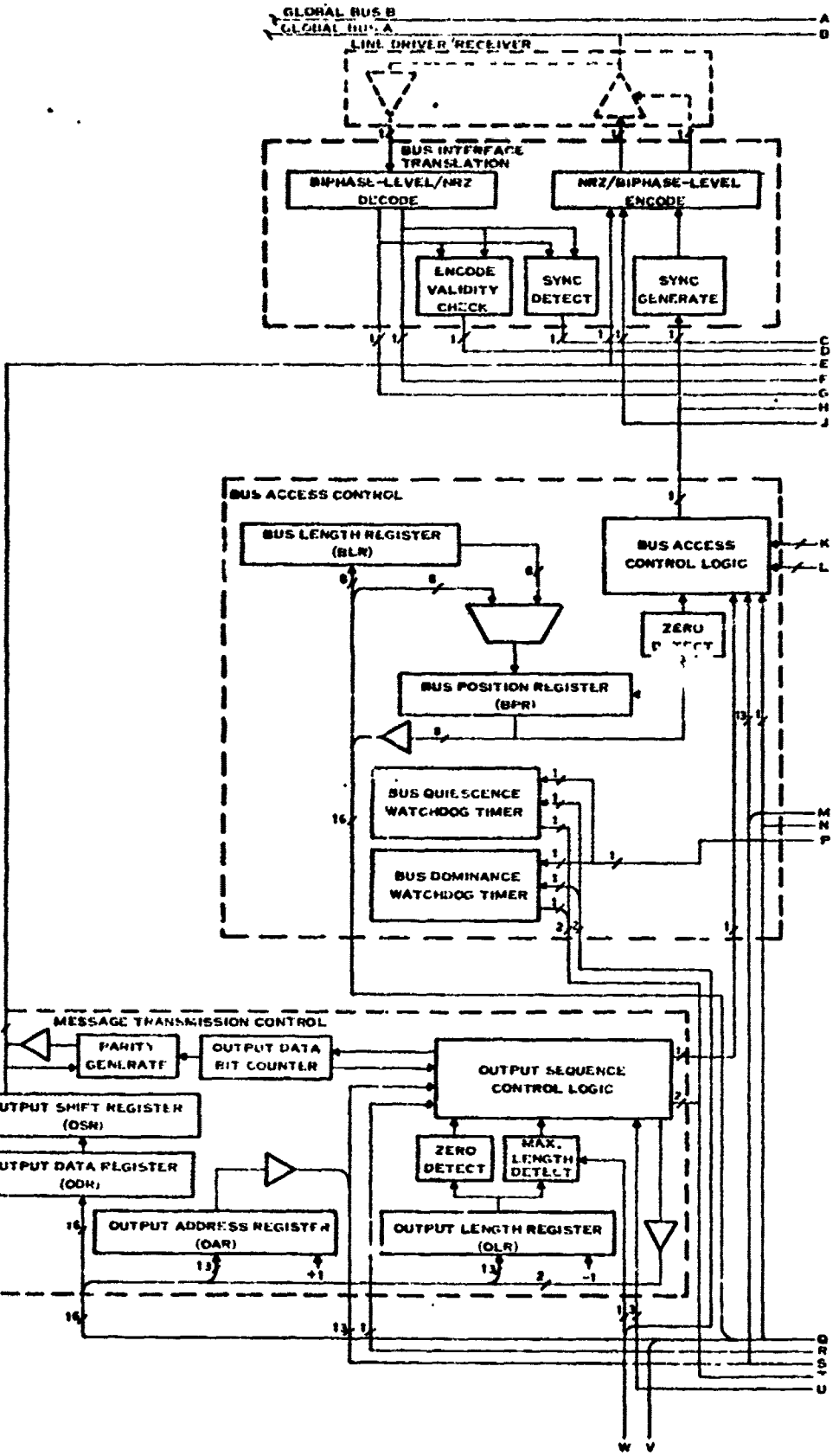
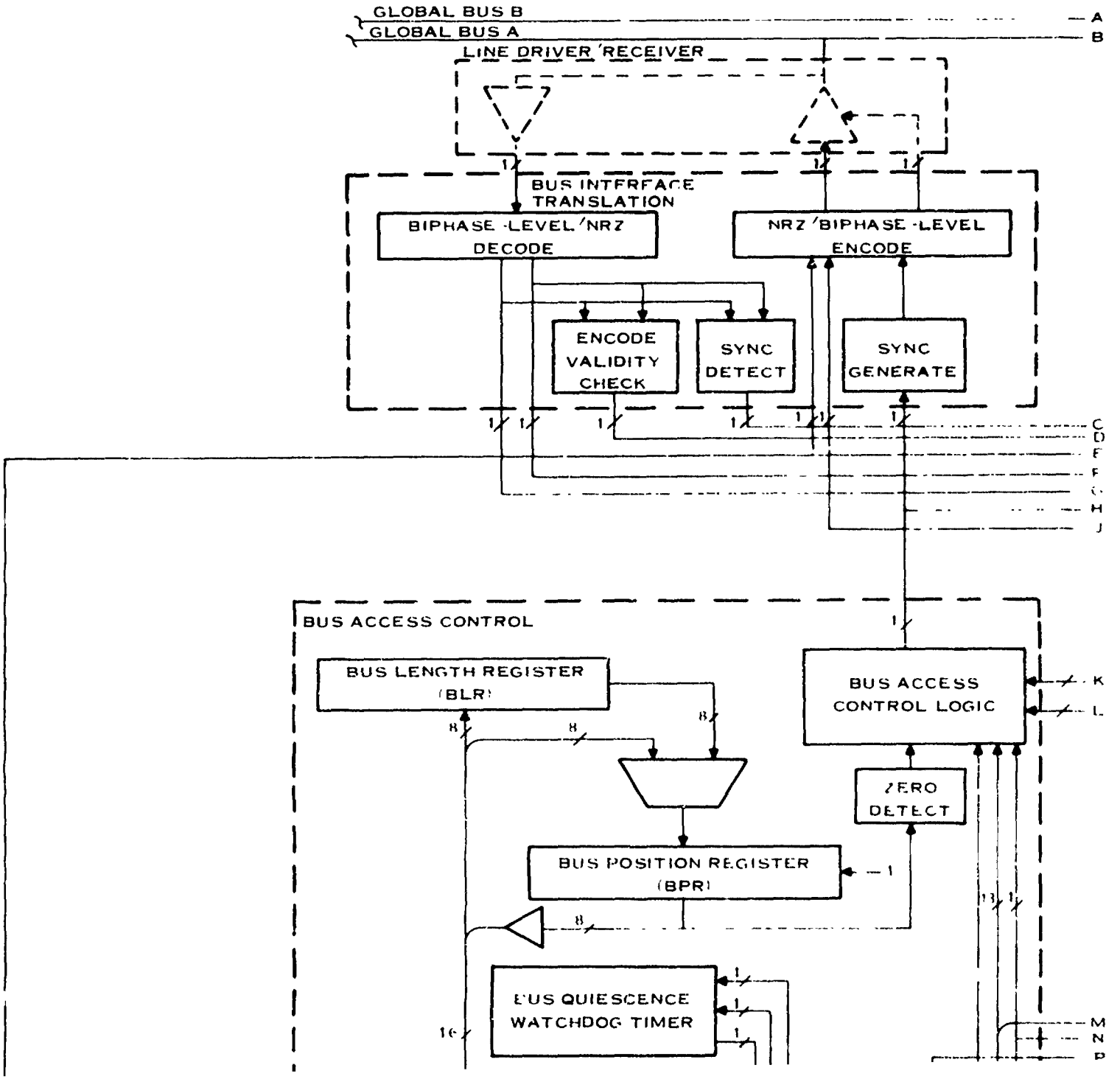



Figure 2b. Global Bus Interface Register-Level Block Diagram (Sheet 1 of 2)


GLOBAL BUS B
GLOBAL BUS A
A
B
LINE DRIVER/RECEIVER
BUS INTERFACE TRANSLATION
BIPHASE-LEVEL/NRZ DECODE
NRZ/BIPHASE-LEVEL ENCODE
ENCODE VALIDITY CHECK
SYNC DETECT
SYNC GENERATE
C
D
E
F
G
H
J
BUS ACCESS CONTROL
BUS LENGTH REGISTER (BLR)
BUS ACCESS CONTROL LOGIC
K
L
ZERO DETECT
BUS POSITION REGISTER (BPR)
BUS QUIESCENCE WATCHDOG TIMER
M
N
P

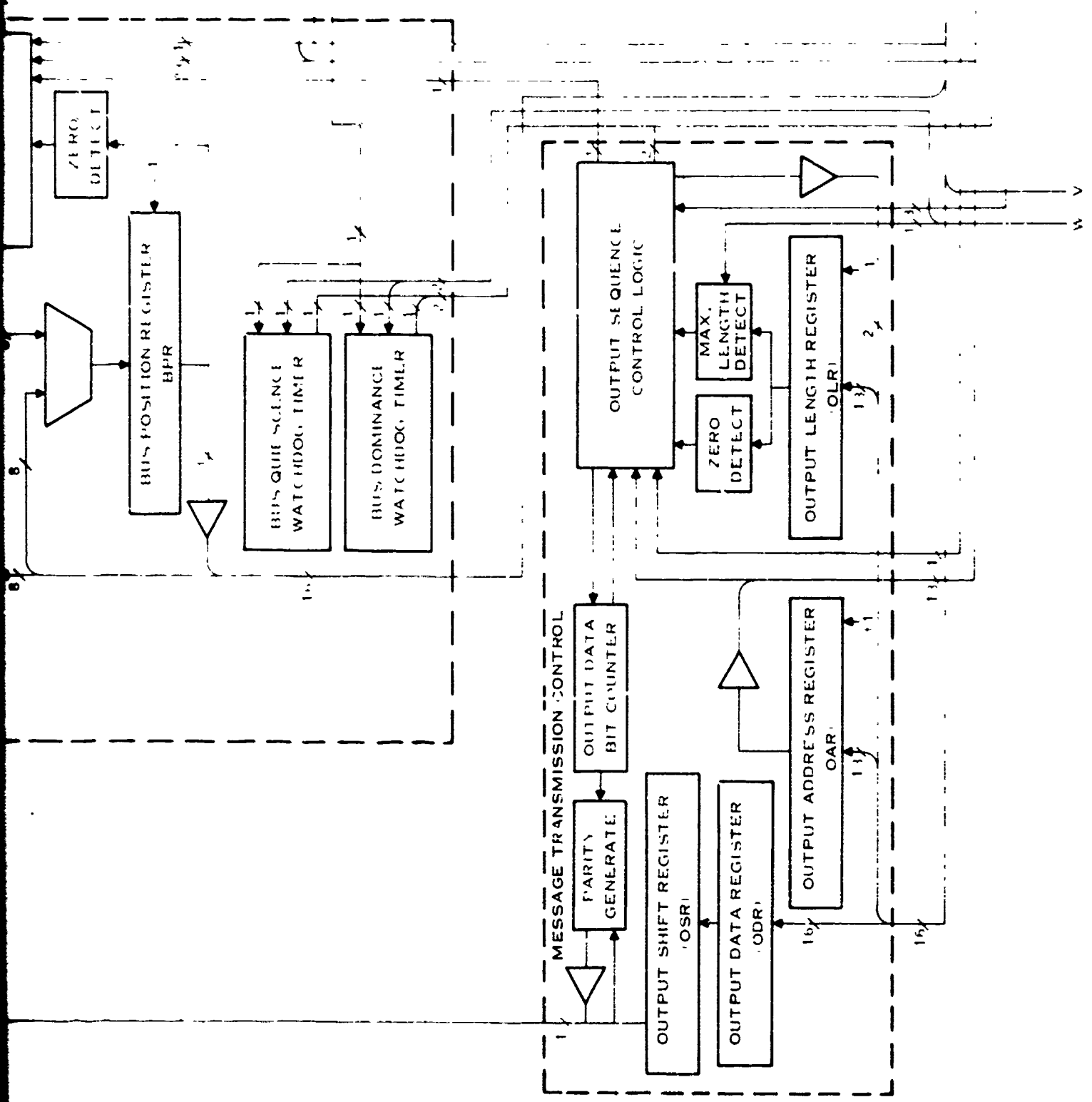
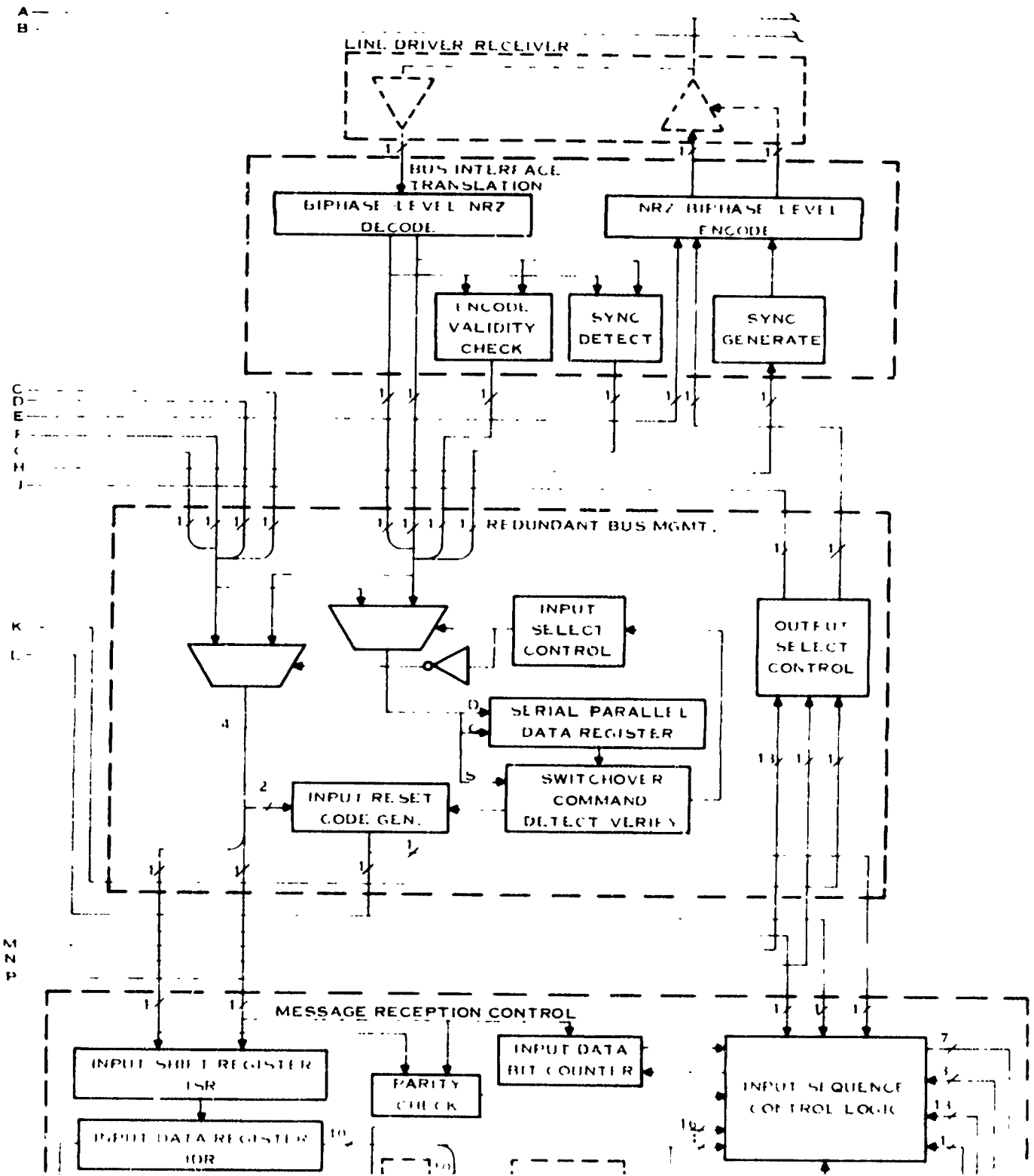



Figure 28 Global Bus Interface Register Level Block Diagram (Sheet 1 of 2)


A
B
LINE DRIVER RECEIVER
BUS INTERFACE TRANSLATION
BIPHASE LEVEL NRZ DECODE
NRZ BIPHASE LEVEL ENCODE
ENCODE VALIDITY CHECK
SYNC DETECT
SYNC GENERATE
C
D
E
F
G
H
J
REDUNDANT BUS MGMT.
K
L
INPUT SELECT CONTROL
OUTPUT SELECT CONTROL
SERIAL PARALLEL DATA REGISTER
INPUT RESET CODE GEN.
SWITCHOVER COMMAND DETECT VERIFY
M
N
P
MESSAGE RECEPTION CONTROL
INPUT SHIFT REGISTER ISR
PARITY CHECK
INPUT DATA BIT COUNTER
INPUT SEQUENCE CONTROL LOGIC
INPUT DATA REGISTER IDR

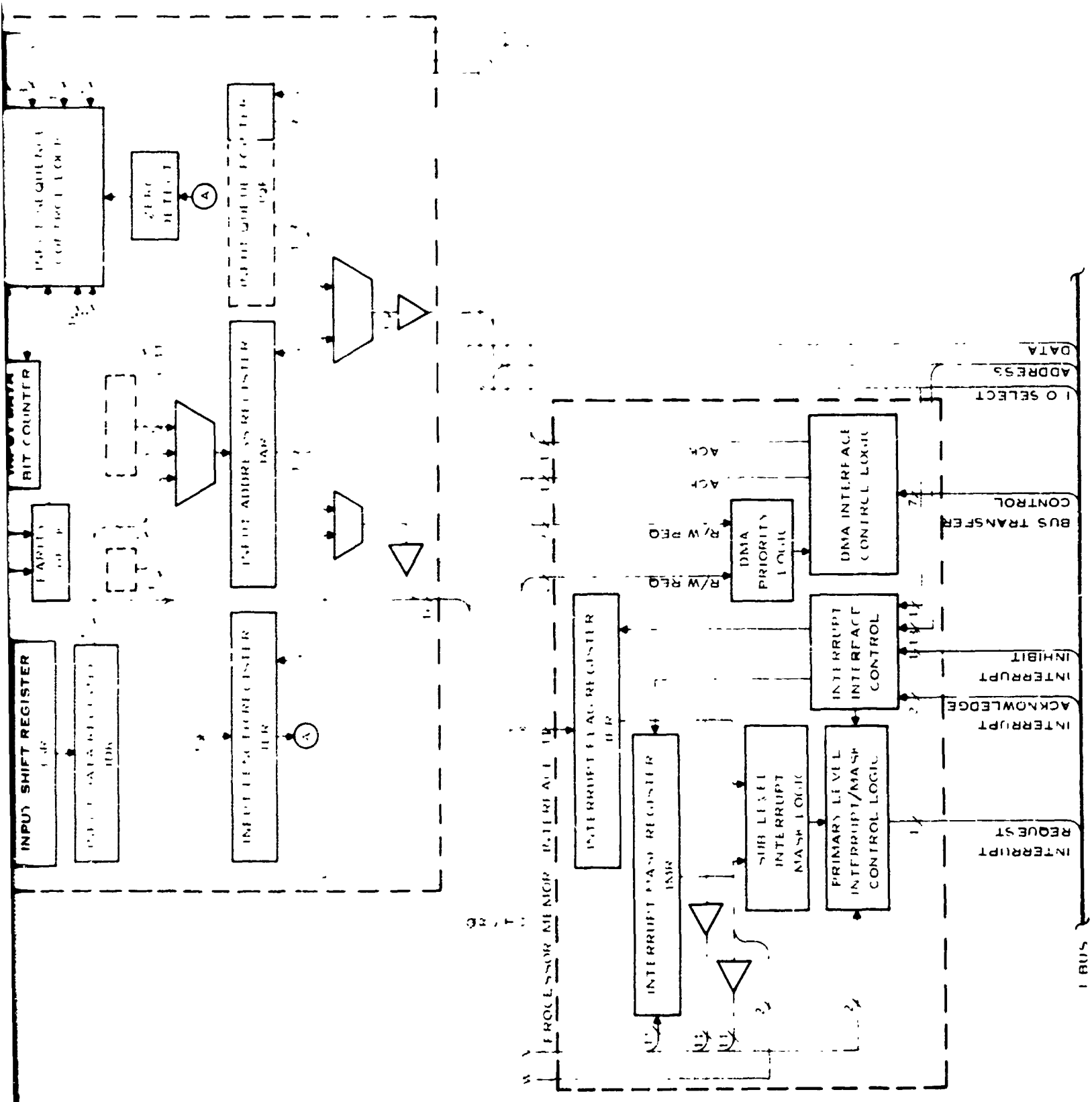



Figure 28. Global Bus Interface Register Level Block Diagram (Sheet 2 of 2)

Preceding page blank

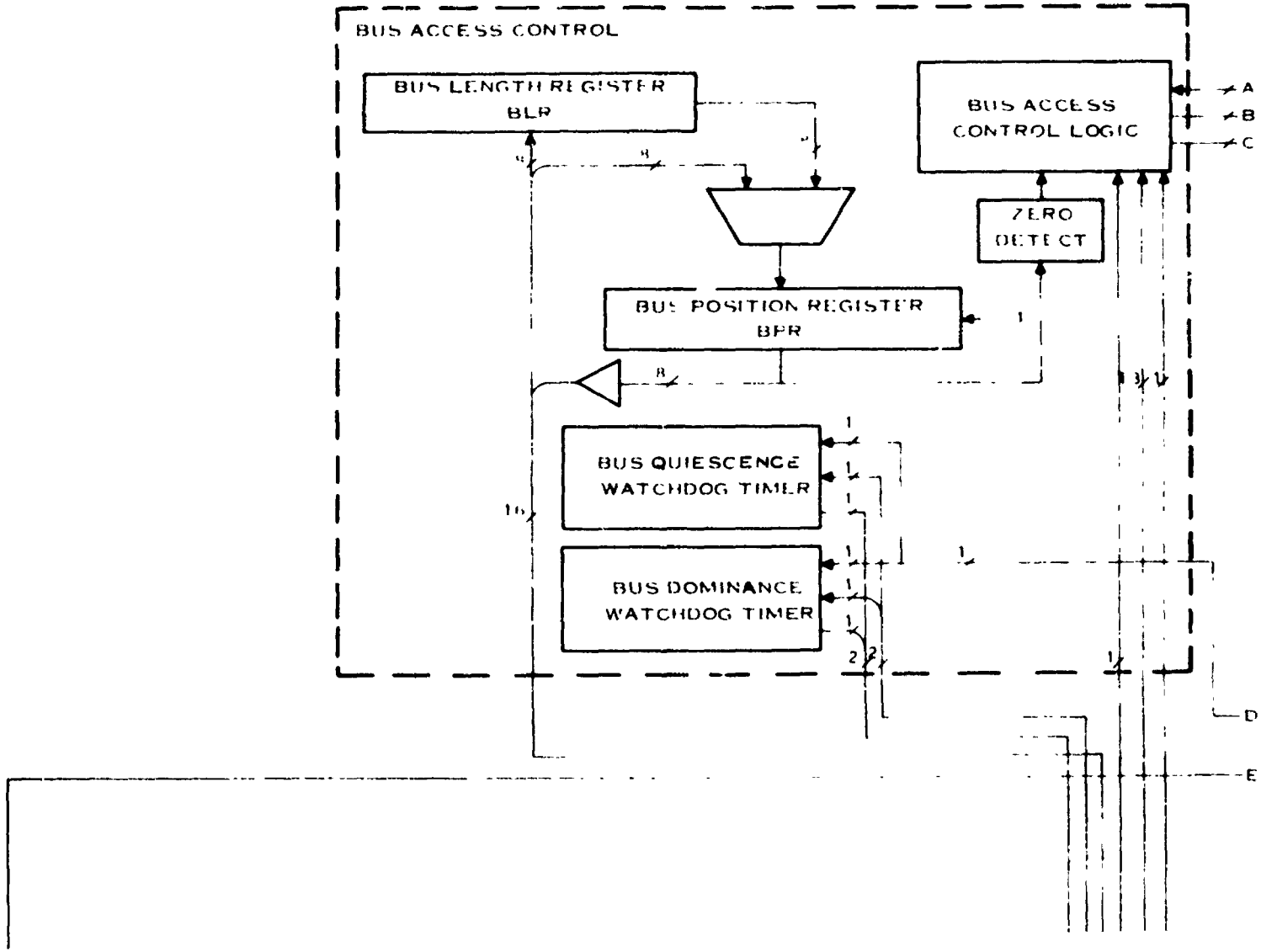
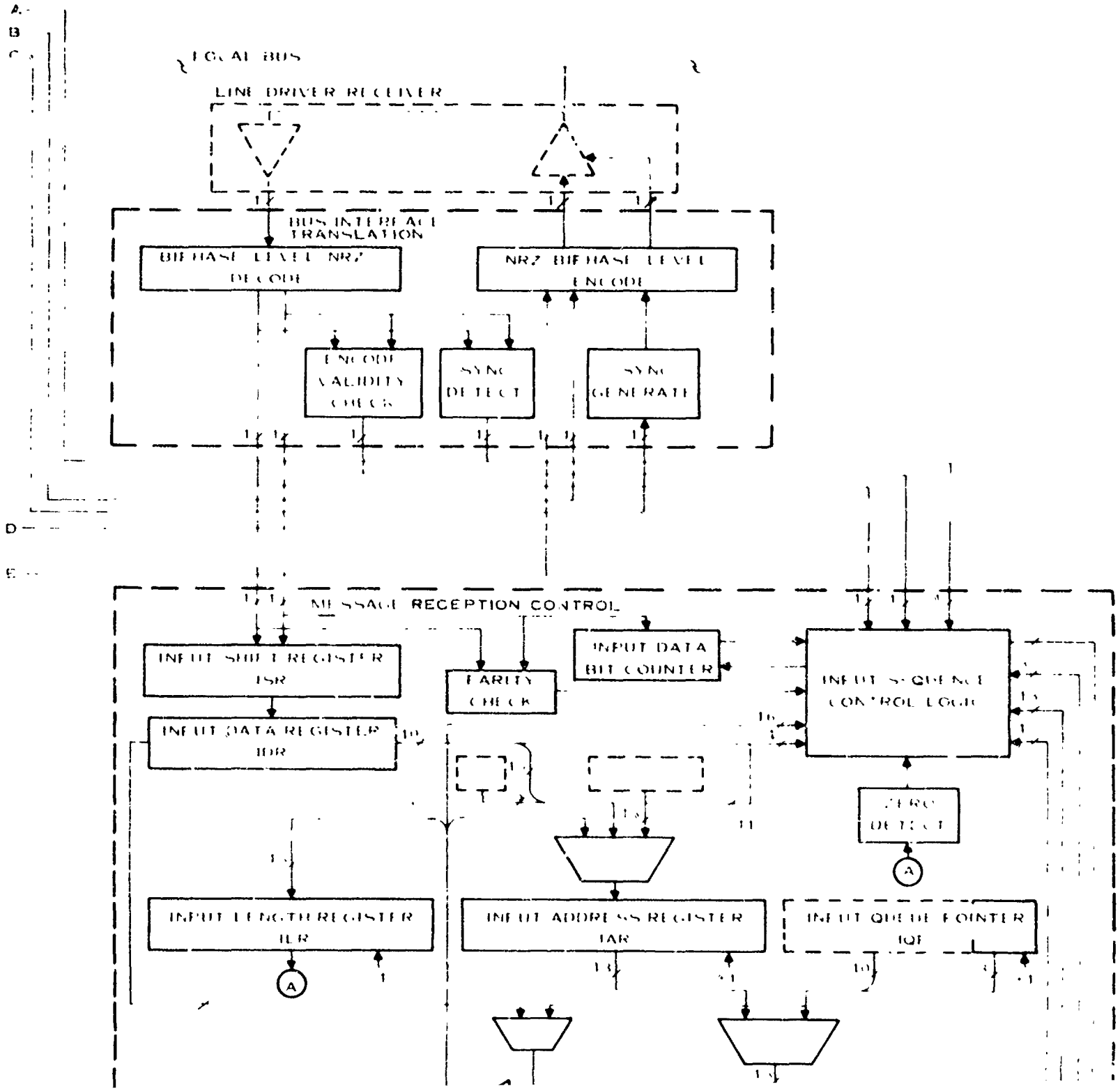

BUS ACCESS CONTROL
BUS LENGTH REGISTER
BLR
BUS ACCESS
CONTROL LOGIC
ZERO
DETECT
BUS POSITION REGISTER
BPR
BUS QUIESCENCE
WATCHDOG TIMER
BUS DOMINANCE
WATCHDOG TIMER
A
B
C
D
E

Figure 29 Local Bus Interface Register Level Block Diagram (Sheet 1 of 2)

Preceding page blank


LINE DRIVER RECEIVER
BUS INTERFACE TRANSLATION
BIPHASE LEVEL NRZ DECODE
NRZ BIPHASE LEVEL ENCODE
ENCODE VALIDITY CHECK
SYNC DETECT
SYNC GENERATE
MESSAGE RECEPTION CONTROL
INPUT SHIFT REGISTER ISR
INPUT DATA REGISTER IDR
PARITY CHECK
INPUT DATA BIT COUNTER
INPUT SEQUENCE CONTROL LOGIC
ZERO DETECT
INPUT LENGTH REGISTER ILR
INPUT ADDRESS REGISTER IAR
INPUT QUEUE POINTER IQP

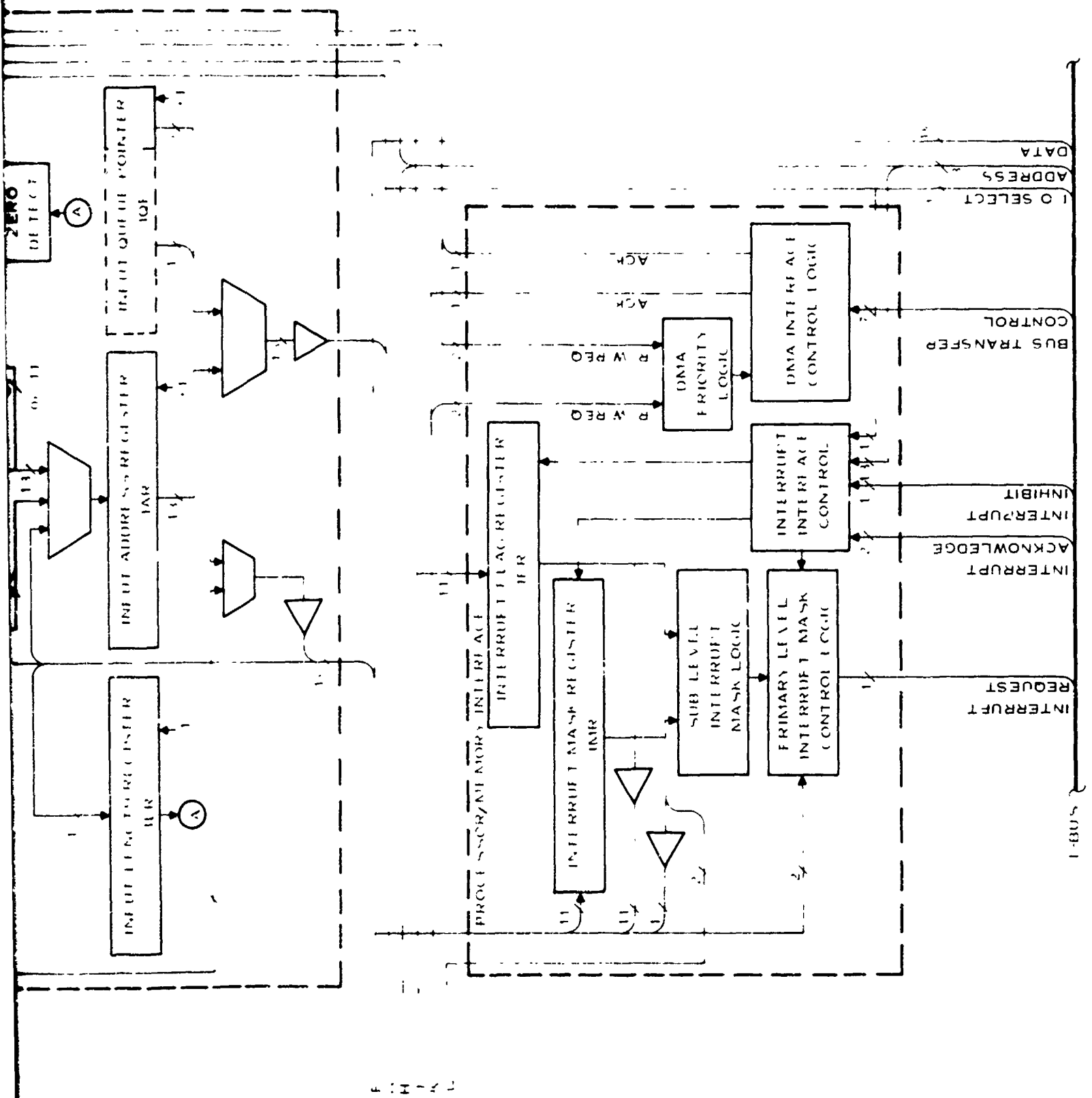



Figure 29. Local Bus Interface Register Level Block Diagram (Sheet 2 of 2)

**Preceding page blank**

**TABLE 6. BUS INTERFACE UNIT**
**FUNCTIONAL HARDWARE COMPLEXITY ESTIMATES**

| | Functional Element | Hardware Complexity (TTL-equivalent gates) |
|---|---|---|
| 1 | Message Output Control | 620 |
| 2 | Message Input Control | 859 |
| 3 | Bus Access Control | 301 |
| 4 | Processor/Memory Interface | 217 |
| 5 | Bus Language Translation | 100 |
| 6 | Redundant Bus Management | 287 |

Total BIU Complexity = 2( 1 ) + 2( 2 ) + 2( 3 ) + 2( 4 ) + 3( 5 ) + 6
= 4,601 gates

the effect of the event on the functional registers of the BIU hardware. The models are driven by exogenous events and also interact with other PE functional element models (e.g., the PE memory) in the performance of particular BIU-related activities.

Basically, the BIU model is segregated into two primary functional activities: Message Input and Message Output. Each major activity is then subdivided into a collection of register-level events associated with the performance of the activity during PE operation. Figures 30 and 31 show the event flow structure of the Message Input and Message Output activities, respectively. Each event model is essentially duplicated for both Global and Local bus activities, with minor deviations occurring in a few event models where G/L operational characteristics differ. The structural and operational characteristics of the PE Functional Simulator are described in Section VII and will not be discussed further herein. It should be mentioned that the modular structuring of the BIU activity model along with each hardware-dependent timing characteristic expressed in parametric form allows flexible definition of the BIU functional operation. The value of this attribute has been proven during BIU design and model development.

The BIU functional design was validation-tested with a test program written in DP/M PE assembly language code and executed by the PE Functional Simulator. This program manipulates a set of exogenous events (predefined messages) defined specifically for testing the BIU behavior under a comprehensive set of test conditions reflecting the inter-PE bus communications environment. The following BIU functional performance parameters and conditions were tested by this program:

All BIU-related I/O instruction executions

Message Reception control

- Message Selection
- Message Length Determination
- Message Buffer Location Determination

**Preceding page blank**

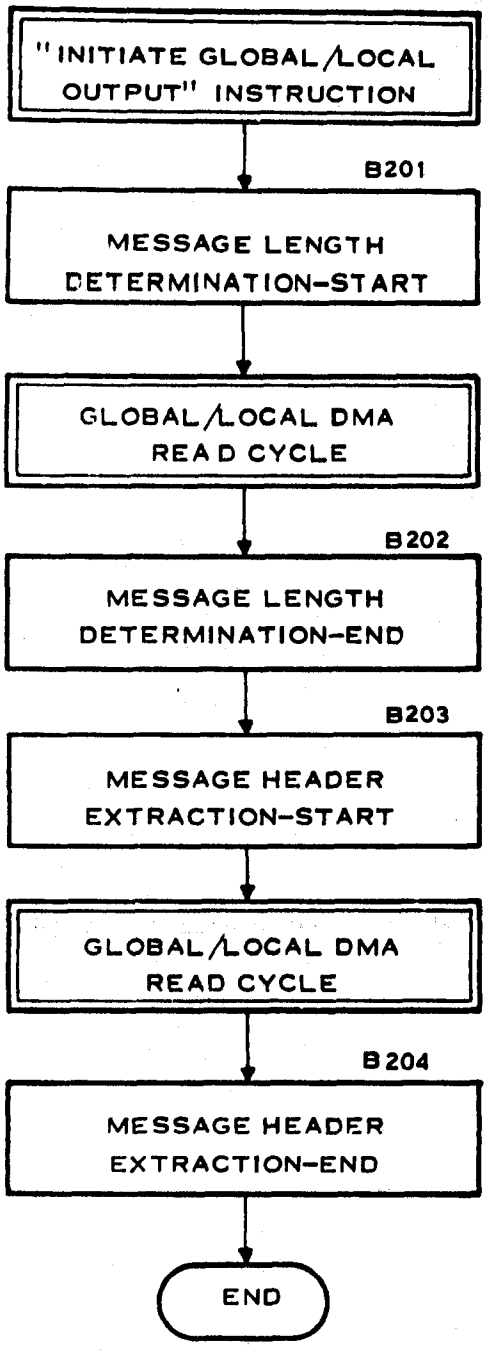



Figure 30. Global/Local Message Output Event Scheduling Flow (Sheet 1 of 2)

Input Message Data Transfer

Input Message Data/Length Error Detection

High-priority Message Recognition

BIU Interrupt Generation (all stimuli)

Bus Access Control Sequencing (including Bus "passing")

Global Output Length Violation Detection/Prevention

Concurrent Global/Local Input/Output activities (all permutations)

Worst-case Input Message Occurrence Timing (back-to-back receptions)

Input Message Queue Posting (with and without message errors)

Bus Quiescent Watchdog Timer

Bus Dominance Watchdog Timer.

This simulation testing has shown satisfactory operational performance characteristics of the BIU functional design (thus making it a viable candidate for use in the DP/M system environment) as well as validating the correct operation and interaction of the various BIU functional elements. Simulation results of particular interest include those which indicate processing time degradation owing to memory cycle stealing effects caused by message reception/transmission. The simulation test cases present an extremely heavy bus loading per unit time condition which may be considered "worse than worst-case" for normal system operation. Even so, it was shown that the interaction of BIU memory cycle stealing activities associated with this rather exorbitant bus loading and program (instruction) execution resulted in a total processing time degradation of approximately only 9 percent. Thus, processing time degradation concerns due to BIU functional characteristics were dispelled. Also of particular interest with respect to BIU design performance were the simulation results which

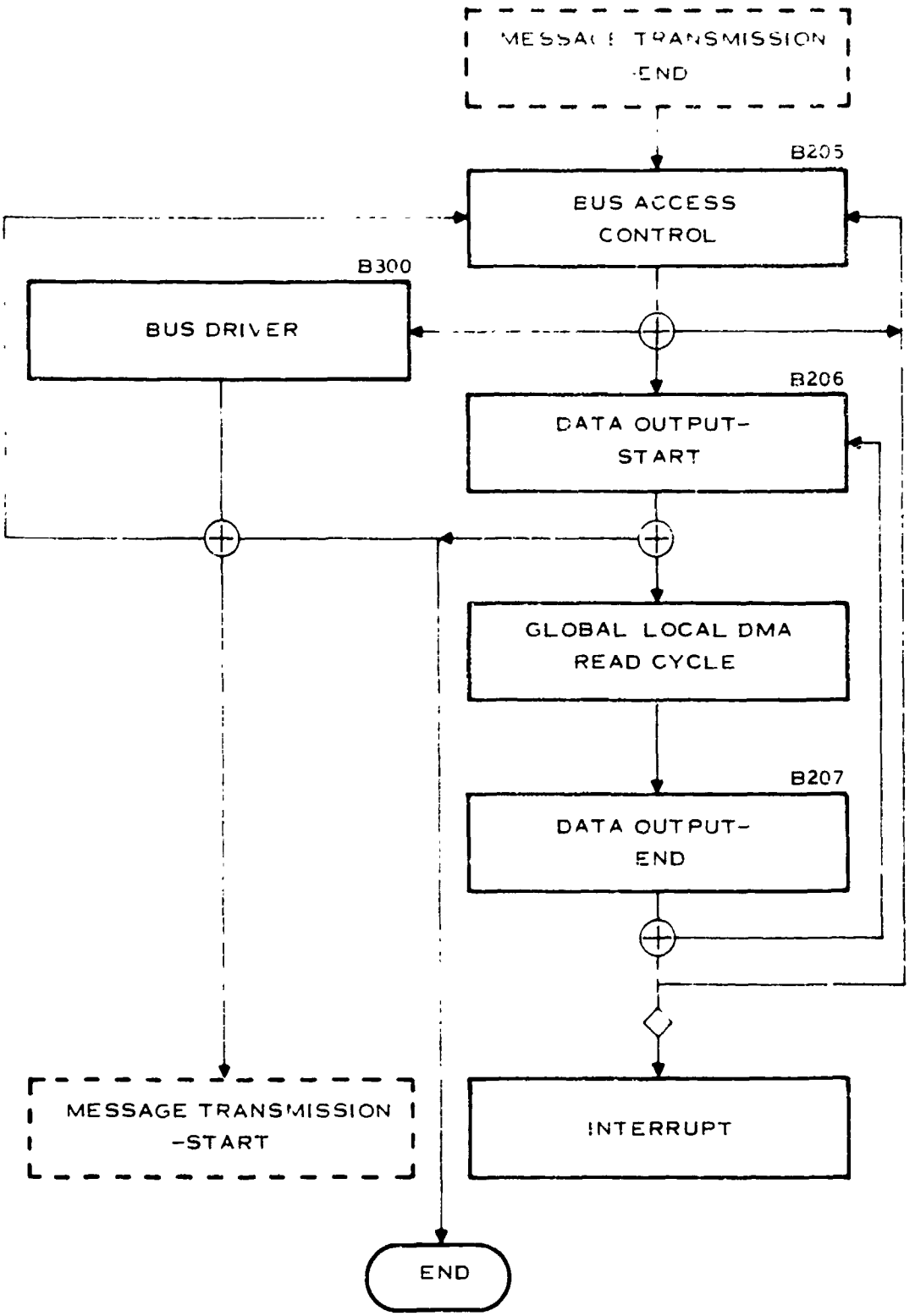



Figure 30 Global Local Message Output Event Scheduling Flow (Sheet 2 of 2)

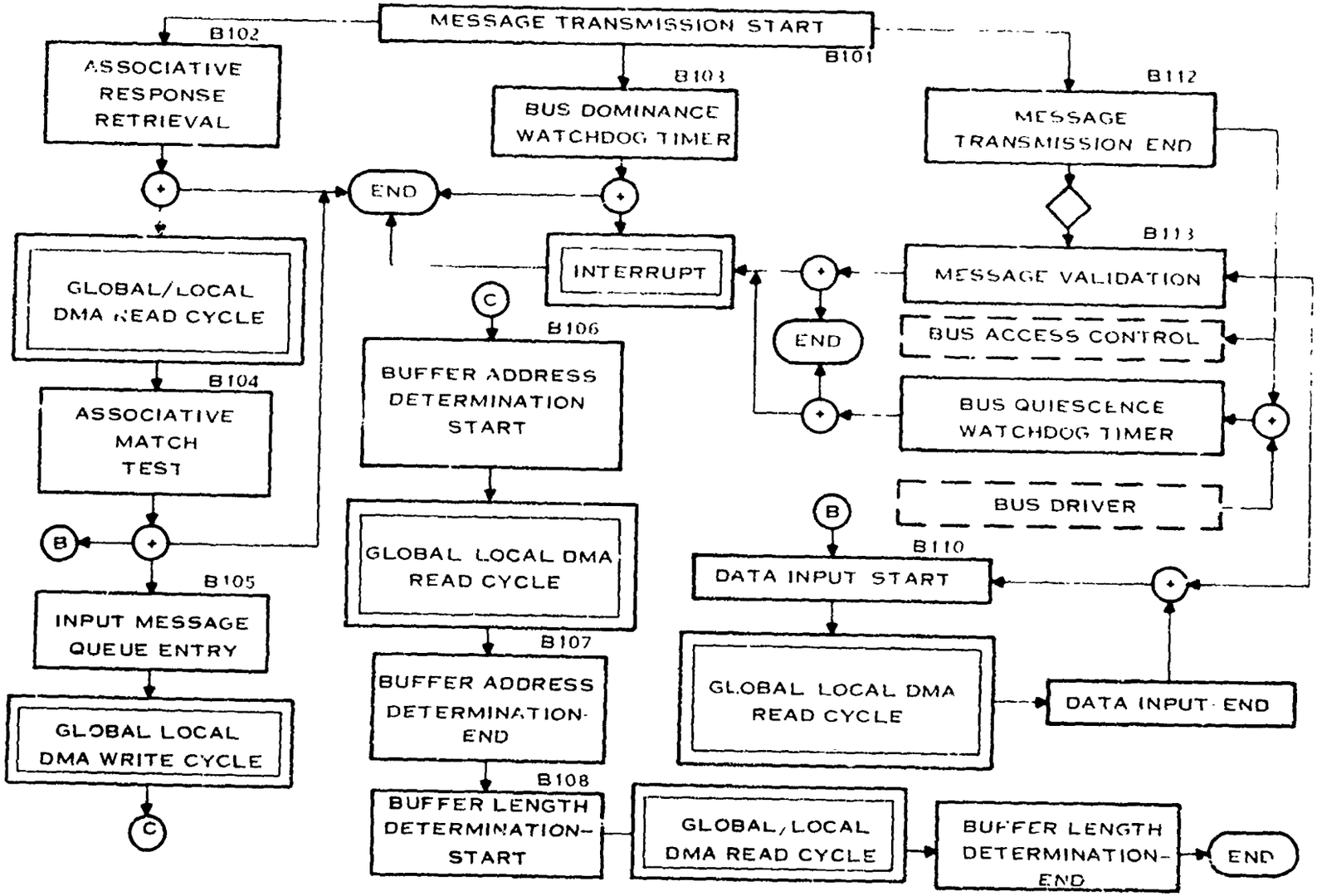



Figure 31. Global/Local Message Input Event Scheduling Flow

showed satisfactory performance of the input message control initialization function under the worst case condition of simultaneous initialization (input buffer setup) procedures in both Global and Local interfaces with attendant memory-cycle stealing conflicts. Worst case input initialization time under this worst case condition (8 microseconds) was shown to be well within that allowed by bus traffic protocol (17 microseconds data word) assuming a minimum memory bandwidth capability of 1 million memory cycles/second.

## G. AREAS OF FURTHER INVESTIGATION

The present Bus Interface Unit design described previously in this section is intended as a tentative baseline approach to a final functional and register level design which may be further tailored to represent a better match to DPM system requirements as these requirements become identified and/or more well-known and understood. It is assumed and recommended that further investigative efforts will use the functional simulation tools provided in conjunction with this design effort to support the design refinement and validation activities.

In the course of formulating the present system design, several areas of analytical tradeoff decisions were made in an effort to minimize hardware complexity and to satisfy system functional requirements with information available at the time of investigation. With proper extended simulation-aided investigative effort, these tentative design parameters may be confirmed or further refined/modified to better meet empirically determined system requirements.

Those functional design parameters which are primary candidates for further analytical and/or empirical investigation include the following:

- Elimination of full-duplex BIU operation: the tradeoff encountered is reduced BIU hardware complexity at the expense of sacrificing BIU "self-test" capability. Investigation should analyze the true value or utility of the self-test feature and the actual quantitative impact of its attendant device complexity on LSI device fabrication.

- A tentative Global message length limit has been chosen at 8 words as an analytical compromise between actual Global message length requirements and bus access latency for the majority of Global transmissions (especially with respect to Global executive functions). Further investigation should not only determine the best-suited quantitative value of this parameter, but should also investigate if requirement exists to justify the inclusion of program-controlled length-limitation to better accommodate perhaps widely varying system application requirements (at the expense of added hardware complexity).

- Queuing of input message identifiers has been arbitrarily fixed at eight-words (i.e., IMQ length of 8 words). The correctness of this quantitative choice can only be verified through extensive simulation of message activity encountered in many various applications. Additionally, the functional integrity of the queueing structure chosen can be examined during the same simulation procedure, and if required, additional queueing operational features can be incorporated into the design (e.g., program-controlled queue-size, hardware queue overflow detection/prevention, etc.). Present design parameters were chosen in a compromise effort to meet analytical requirements with minimum hardware complexity. However, further empirical investigation may reveal a

requirement for added functional capabilities at the expense of BIU hardware complexity

- BIU function memory allocation required by the present design uses fixed (hardwired) portions of PI memory. Actual usage experience via extended situation may reveal a desirability to allow program control or selection of memory areas allocated to BIU functions (Again at the expense of added BIU hardware complexity.)
- Actual or unique prioritization requirements of Global Local bus users may exceed those accommodated by the baseline bus control algorithm and or supercommutation capability. If such is the case, it can be revealed only through empirical investigation
- Any degrading influences on PI processing performance caused by the BIU's functional usage of PI memory can be exposed only by exercising actual applications code (programs) via simulation. If any unexpected but significant degrading influences are found, it may be decided to change the physical design of the BIU by providing a separate, unique memory to provide the associative memory functions required in the design of the message input control element

Once the actual costs of the BIU hardware devices are ascertained, it may be desirable to adopt an alternate technique of redundant global bus management. The present design recommendation is a result of stressing overall BIU hardware simplicity. However, if BIU hardware costs are significantly small enough with respect to overall DPMS system cost objectives, it may be desirable from the viewpoint of application flexibility and extendability, to implement redundant bus management with an individual BIU-per-bus approach. This approach would essentially require replication of the functional hardware required to interface a given bus to the PI for each bus required in the system (e.g. three BIUs would be required for the baseline system) with the elimination of the requirement for a unique Redundant Bus Management function. This approach offers little if any functional impact on system design operation. System level redundant bus management techniques would not require any alteration. The only functional impact on BIU operation is the possible requirement for adding programmable enabling disabling of individual bus interface message reception activity after a faulty bus is detected and isolated. The primary physical implementation impact is that the BIU must be modularly partitioned to allow a "BIU-per-bus" system construction. In addition, each BIU would require a "programmable" identity capability for all characterizations of each individual bus interface (e.g. PI memory allocation for separate input message queues, interrupt assignments, etc.) during system initialization, thus introducing additional hardware complexity to the BIU design. However, if properly implemented, the approach may realize added flexibility benefits by permitting a variable number of system buses per application with a greater number of topological configurations allowed.

It should be emphasized here that the above identified candidate areas of further design investigation via accumulation of empirical data (by simulation) do not necessarily suggest functional design deficiencies in the present baseline BIU design. They do, however relate to areas of design tradeoffs recognized in the present functional design process with tradeoff decisions made primarily in favor of minimum hardware requirements, based on analytical investigation with little, if any, empirical data available to aid in the decisions. It is thus recommended that the next phase of design validation, i.e., validation with empirical data gathering via application simulation experiments, be performed before final BIU design formulation and acceptance.

# SECTION IV
# PROCESSOR/MEMORY DESIGN

This section summarizes the recommended design specifications for the DP M PE processor and memory modules. An internal PE bus facility (the I-BUS) is described as is its potential use for aiding hardware and software maintenance functions. The design approach for these modules was guided by a number of objectives, with primary emphasis on identifying and incorporating those computational features required for a large segment of different avionic processing system applications. Likewise, a parallel effort was made to stress simplicity, modularity, standardization of interfaces, and low-risk hardware features that would result in a cost-effective design and implementation. The design effort had as its goal the definition of a standard reprogrammable PE which was:

Computationally capable so as to be suitable for a wide range of avionic processing problems

Simple enough for the processor to be fabricated as a low-cost module.

These requirements were reviewed in light of previous Air Force avionic processor requirements/analysis efforts and current trends in commercially available technologies and design concepts. The general PE characteristics satisfying the first requirement are

16-bit processor architecture

250 KIP (thousand instructions per second) processor performance

8K word memory module per PE.

The second requirement for a low-cost implementation appears attainable based on existing semiconductor technologies and their current expected rate of advancement, and is discussed in Section VI.

Items considered in the design specification of the processor/memory were:

Elimination of the need for instruction code modification at run time (e.g., dynamic shift counts)

Addition of a short instruction format for high-usage instructions to conserve program space and to improve execution speed

Inclusion of certain uninterruptible basic read/modify/write instructions (clear bit, set bit, test and set bit, and store through mask) to prevent incorrect action as the result of an interrupt

Identification of a need for signed multiply and divide for numerical data processing

Definition of a common inter-PE data bus (I-BUS) to provide for internal communications between the Processor, Memory, BIU and I/O modules.

Investigation of a simplified Processing Element maintenance control panel and AGE concept implementable through the use of the I-BUS.

## A. PROCESSOR FUNCTIONAL ARCHITECTURE

The following paragraphs describe the functional architecture of the DP M processor. The term "functional architecture" is used to describe that set of computer capabilities available to

the programmer and is exemplified by the programmable registers, data types, and instruction set. The design of the DP M processor followed a top down approach, i.e., the design started with a review of existing and projected digital avionic processing applications and identified various alternatives to be candidates for incorporation into the DP M processor design. The material presented in this subsection is an overview of the functional architecture of the processor and some of the design tradeoffs identified during the design effort. The items that are summarized include the processor organization, instruction complement, addressable registers, and data types. The results of sample instruction usage analysis are presented as are the interrupt and initialization requirements for the processor.

The DP M processor is a 16-bit, two's complement, eight register file architecture. The processor provides the necessary functions to perform arithmetic, logical, shift, and data transfer operations. Operands for the execution of instructions are obtainable from the register file and program memory and results are placed into either the register file, program memory, processor status word, or transferred as input output data.

### 1. Addressable Registers

The DP M PE processor has an eight-register general register file with two registers designated for the program counter (PC) and the interrupt stack pointer (ISP). These general registers can be used as accumulators, indexes, bases, and or to hold temporary variables. This register file configuration was selected after a careful review analysis of alternative approaches to the addressable register configurations.

The alternatives examined for the addressable registers included both accumulator register file organizations. Many processor designs have used an accumulator and extended accumulator, with and without additional index registers. A more recent approach is to have a file of up to 16 general-purpose registers. These registers can be used as accumulators, index registers, and/or to hold temporary results. The advantages of the dedicated accumulator approach are:

Minimizes Hardware. With a general register file, there are more registers and several multiplexers that are not needed with a simple accumulator extended accumulator configuration. In addition, when a register file is used, there frequently will also have to be additional working registers used in conjunction with the register file. These working registers are simply the micro-programmer's accumulator and extended accumulator (or MQ register). It is possible though, to implement the general register file as a part of main memory, at the expense of decreased performance due to the bandwidth limitation of memory for both storage and file operations.

Conserves Instruction Bits. If a general register approach is used, bits must be reserved in the instruction word to specify which of the general register(s) is to be used. In the simple accumulator extended accumulator approach, there are few if any bits used to specify registers.

Minimized Interrupt Save Restore. The more registers that exist, the longer the saving and restoring of them takes when entering and returning from an interrupt service routine. Thus, the accumulator/extended accumulator architecture tends to be faster and more efficient at interrupt servicing. One point in favor of the virtual register file is the ability to switch quickly to a new set of "registers" by merely assigning an alternate section of memory to be the new file.

The disadvantages of the simple accumulator/extended accumulator architecture are

Intermediate Loads Store Since there is only one accumulator extended accumulator, intermediate results and temporary values must be retained in main memory. This causes a significant increase in the number of load and store instructions. The result is an increase in program size and execution time.

Extra Index Instructions With the simple accumulator extended accumulator configuration, extra instructions are needed to manipulate the index register(s). With the register file approach, any and all instructions can be used to manipulate the indexes.

As hardware costs have decreased, the performance and efficiency of the register file architecture have become most attractive. For the DP M processor, the two primary advantages of a register file are.

Reduced Memory For avionics application software there are typically 4 to 5 times as much memory devoted to program as data Thus, there can be a significant sevings in memory due to eliminating many of the instructions associated with loading and storing temporary values and intermediate results.

Simplified Control Structure There is no distinction between index register operations and other normal register computations. Thus, there are fewer instruction operations to decode and execute. This causes the control structure to be smaller and simpler

The selection of a general register file organization required several additional design decisions. The number of registers in the file and the functional use of the registers by the processor instruction set and architecture had to be determined. The determination of the number of registers had several factors to consider One opinion always persists: most programmers and compiler writers feel there are never enough registers and/or memory for real-time applications. Thus the difficult problem is one of determining what the majority of the avionic processing applications require. Because of hardware, instruction bits used per instruction, and interrupt save/restore considerations, the number of general registers needs to be held to a usable convenient minimum. Processors have been built with 4, 8, and 16 general registers. Most contemporary minicomputers use eight general registers, and avionic programming experience indicates that the majority of programs generally require 4 to 8 accumulators, indexes, and temporary storage locations. For these reasons, the DP M processor was specified with an eight-general-register file, with two of the registers having special functions.

The next design consideration was the handling of the processor program counter (PC). The PC is useful as a base to address into the program space both for branching and fetching fixed data. It also simplifies the manipulating of the PC if it is one of the general registers. This allows both the PC to be used as a base and the use of the full instruction set to manipulate it. For example, an unconditional branch is a register (program counter) load and a branch relative can be effected with a register-add instruction.

A second file register is designated for use as a system or interrupt stack pointer (see Subsection IV.A.6 for the processor interrupt system design). The disadvantage of including the stack pointer in the file is that this uses up one more of the general registers. This is only true, though, if there is not re-entrant programming. If there is any re-entrant programming, a stack

and a pointer are a necessity. For the DP/M PE, it was decided that the advantages of using one of the general registers for a stack pointer outweighed the disadvantages. Thus, for the DP/M PE, general register 6 is dedicated to being the interrupt stack pointer. It can also be used as a run time dynamic memory stack pointer where multiple registers can be saved/restored automatically in available memory for subroutine and interrupt processing routines.

2. Processor Word Length

The basic word size for the DP/M PE has been set at 16 bits. The choice of a 16-bit addressable word was made in view of the following data:

A large portion of the avionic sensor/actuator input and output signals have an accuracy range from 10 to 12 bits, with a relatively few signals whose digital value exceeds 16 bits.

Most numerical calculations with this sensor data can be accomplished with 16-bit fixed-point arithmetic, with some instances requiring double precision (32-bit) operations.

The use of a 16-bit address field within a PE instruction format permits an instruction address reach of 64K words of memory, which is more than adequate for expected DP/M applications.

Characters and short integers (−128 to +127) can be packed two per 16-bit word, while double words (32 bits) are more than adequate for double precision fixed-point variables and for software floating-point variables.

An additional consideration in the basic word size selection was the number of bits needed for instructions. This is particularly important since, except for large data base programs, there is normally 4 to 5 times as much memory devoted to instruction storage as to data. Various studies have been made as to the information content of instructions for different machines (IBM 7090) and have shown that an average number of useful bits for instruction field is in the 15- to 17-bit range. Thus, if the instruction set allows for variable-length operations (i.e., short one word and long multiple word), then a 16-bit word length appears suitable for instructions also. This is especially true if special attention is given to the most frequently executed instructions. As will be seen later in this section, this choice was quite satisfactory since 75 percent of the DP/M executive code and 64 percent of the diagnostic code were able to use single word (16-bit instructions.

Two other factors that were considered in the selection of the 16-bit word length were the likely commercial market evaluation of LSI memory and micro processor devices. Present indications concerning semiconductor memory devices that either currently exist or are projected for development indicate device partitioning in multiples of 2 or 4 bits and a 16-bit data word could efficiently use either approach. The 1972–1974 product boom of 4- and 8-bit microprocessor chip sets lends creditability to the DP/M concept of a low-cost LSI commercial equivalent processor suitable for 1978–80 avionic system applications. It is felt that as LSI technologies improve, the 16-bit "micro class" processor will be an established computing device within the next 2 years that will be suitable for use in DP/M-like systems.

## 3. Data Types

The DP/M processor data types available to the programmer are bits, bytes (characters and short integers), and integers (single and multiple precision). Associated with each data type is a class of instructions. As with most processors, the single precision integer processing is used for address generation.

### *a. Bits*

Bits are addressed by specifying the whole word that contains the desired bit(s) and then specifying the bit or bits within the word either with a mask and using an AND or OR operation, or by specifying the bit position within the word and using a SET, CLEAR, or TEST BIT operation.

### *b. Bytes*

A byte is used to hold a character or a short integer (Figure 32). The DP/M byte is an 8-bit two's complement number which can take on the values -128 to +127. Before a byte is used (Figure 33) it is sign extended into a 16-bit quantity.

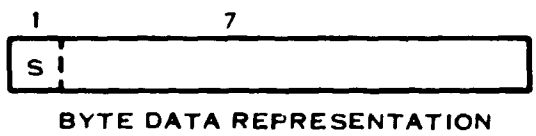



Figure 32. BYTE Data Representation

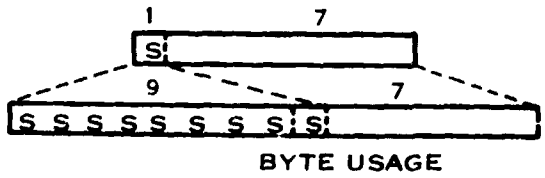



Figure 33. BYTE Usage

### *c. Single Precision Numbers*

Whole words are used to store addresses and single precision numbers (integer or fixed point). A single precision number is a 16-bit two's complement number which as an integer can take on the values -32,768 to +32,767.

### *d. Multiple Precision Numbers*

The format selected for multiple precision numbers (integers or fixed point), as shown in Figure 34, was derived in order to simplify the software necessary to implement the basic arithmetic functions of add, subtract, multiply, and divide. Thus the most significant part will be in the first word and the least significant in the last word.

## 4. Instruction Complement Selection

The selection of an instruction complement was based upon several factors. A review of many of the contemporary commercial and airborne minicomputer instruction sets reveals a

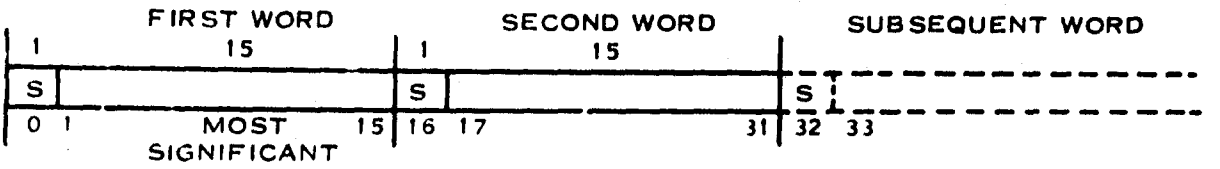



**Figure 34. Multiple Precision Data Representation**

striking similarity in the capabilities offered by the different designs. The approach on DP/M was to analyze available existing avionic operational software, determine the set of operations most frequently required of real-time avionic programs, and define an instruction set that reflects these requirements. In addition, a concentrated effort was made to simplify the programmer's use of the instruction set and the hardware control for decoding instruction formats. The selection of the DP/M instruction formats stressed minimization of decode control and standardization of the instruction operand derivation and execution cycles as much as possible.

Every DP/M instruction execution cycle can be viewed as occurring in two phases. The first phase is associated with obtaining an operand (from memory, a register, or within the instruction), and the second phase is the use of that operand in conjunction with a second operand (a register or memory location) to effect the desired operation. This process of an instruction specifying two functions (the selection of operand sources and the instruction execution cycle using these operands) has two major advantages. The software programmer (or compiler) has a basic set of instructions that can be used with a variety of source/destination operands. For example, an add instruction can have an implied destination register and multiple operand sources (another register, contents of a memory address that may be indexed, or an immediate data value imbedded within the instruction). This flexibility of specifying a wide variety of operand types not only adds power to the use of the instruction set, it also simplifies the control logic required to implement a given set of instructions. Whenever an instruction is fetched from memory and ready for decode, a predefined set of fields within the instruction can be decoded to derive the necessary operands. This operand derivation can be independent of the instruction execution cycle and, hence, become common to all operations. Once the operands have been derived and placed in the appropriate internal working registers, one common instruction execution cycle can be invoked to perform the operation using the internal working registers for operands.

This two-step approach of operand derivation and instruction execution is further enhanced in the DP/M processor design by the use of a minimum number of different instruction formats. This emphasis on regularity is also intended to simplify instruction decode and encourage efficiency of instruction usage and execution. Two instruction formats have been defined for the DP/M.

- The standard instruction format which provides both short (one-word) and long (two-word) instructions
- The extended short instruction format which provides a limited number of short instructions for high frequency/efficiency operations.

The standard format provides a full complement of instructions with the desired types of operands determining the length of the instruction word generated. Operands found in registers require short instructions while operands requiring full memory addresses or 16-bit or constant

data require long instructions. The extended short format provides small constant (immediate) values to be used for either data and or displacements in one short instruction to minimize program memory and execution time for high-frequency operations.

### *a. Standard Format Instructions*

The standard format instructions are intended to provide a full range of operations suitable for real-time avionic system applications like DP M. A full set of instruction operations and operand modifications were considered for incorporation into the DP M processor specification. The operand types and addressing modes initially identified as candidates for DP M were

Constant (or immediate) data that is imbedded as part of the instruction format

Register-to-register where both operands are located within registers in the file

Direct where the operand is the contents of a memory address in the instruction

Direct indexed where the operand is the contents of a memory address whose value is the sum of a field in the instruction and an index register

Indirect where the address field in the instruction is a memory location whose contents are the memory address of the desired quantity

Indirect Indexed is identical to Indirect except indexing is applied

Register Indirect (RI) where the memory address of the desired operand is contained in one of the file registers

Register Indirect with Autoincrementing (RIA) is identical to Register Indirect except the register containing the memory address is incremented automatically after the operand is fetched.

The latter two forms (RI and RIA) of indirect addressing use the contents of a register as an address for the data. These two forms of addressing allow efficient data accessing within processing loops, and the RIA form is particularly efficient for stepping sequentially through tables. Register Indirect is used for arbitrarily generated addresses. By examining several samples of avionic applications code, it was observed that (1) there was frequent use of direct indexed instruction (the fourth item above) with an address displacement field of zero (i.e., the address was contained in the register), and (2) table access within loops was usually sequential. These observations influenced in part the selection of the following DP M processor addressing operand modes

Register-to-Register (RR)

Register Indirect (RI)

Register Indirect with Autoincrement (RIA)

Constant or Immediate Data (C)

Direct (D)

Direct Indexed (DX)

The implementation of these six desirable addressing modes was accomplished in the following manner. The RR, RI, and RIA modes were given unique hardware description codes for the operand derivation cycle, as was the Direct memory address modification. The remaining two addressing modes, Direct Indexed and Constant, were implemented without unique operand modification descriptors. The use of one of the register file members for a default condition of

no indexing allows the Direct modification to be non-indexed (D) or indexed (DX), depending upon the value of the index register. A standard convention commonly used is that register zero (R0) is non-indexable, and that a non-zero index register implies automatic indexing (i.e., DX is actually a D modification with a non-zero index register). This convention is the approach used for the DPM standard format direct/direct indexed instructions as well as the memory input/output instructions. The constant modification provides 16 bits of immediate data for use by the instruction. This modification is easily accomplished by using the register file member containing the program counter (PC) for the R1A operand register (i.e., the PC points to the next 16-bit word and is automatically incremented to the next instruction after the constant data has been fetched).

**TABLE 7. STANDARD FORMAT INSTRUCTION SET**

| | | Instruction Fields | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0 | | | | 15 | 16 | 31 |
| | Mnemonic | XX | CCCCCC | MM | TTT | RRR | AAAAAAAAAAAAAAAA | |
| **Data Transfer** | | | | | | | | |
| LOAD | L | 00 | 000001 | | | | | |
| STORE | ST | 00 | 000010 | | | | | |
| STORE THROUGH MASK | STTM | 00 | 010000 | | | | | |
| PUSH | PSH | 00 | 010001 | | | | | |
| MOVE AND AUTOINCREMENT | MVA | 00 | 010010 | | | | | |
| PUSH MULTIPLE | PSHM | 00 | 110001 | | | | | |
| POP MULTIPLE | POPM | 00 | 110010 | | | | | |
| **Arithmetic** | | | | | | | | |
| LOAD TWO'S COMPLEMENT | LTC | 00 | 010011 | | | | | |
| ADD | A | 00 | 000011 | | | | | |
| SUBTRACT | S | 00 | 000100 | | | | | |
| COMPARE SIGNED | C | 00 | 000101 | | | | | |
| MULTIPY | M | 00 | 000110 | | | | | |
| DIVIDE | DV | 00 | 000111 | | | | | |
| **Logical** | | | | | | | | |
| LOAD ONE'S COMPLEMENT | LOC | 00 | 001000 | | | | | |
| AND | N | 00 | 001001 | | | | | |
| OR | O | 00 | 001010 | | | | | |
| EXCLUSIVE OR | XO | 00 | 001011 | | | | | |
| **Bit** | | | | | | | | |
| SET BIT UPPER BYTE | SBU | 00 | 010111 | | | | | |
| SET BIT LOWER BYTE | SBL | 00 | 011000 | | | | | |
| CLEAR BIT UPPER BYTE | CBU | 00 | 011001 | | | | | |
| CLEAR BIT LOWER BYTE | CBL | 00 | 011010 | | | | | |
| TEST BIT UPPER BYTE | TBU | 00 | 011011 | | | | | |
| TEST BIT LOWER BYTE | TBL | 00 | 011100 | | | | | |
| **Shift** | | | | | | | | |
| SHIFT SINGLE | SFTS | 00 | 001100 | | | | | |
| SHIFT DOUBLE | SFTD | 00 | 001101 | | | | | |
| **Program and Interrupt Control** | | | | | | | | |
| BRANCH ON CONDITION | BC | 00 | 001110 | | | | | |
| EXCHANGE STATUS AND PC | XSPC | 00 | 001111 | | | | | |
| RETURN FROM INTERRUPT | RINT | 00 | 100010 | | | | | |
| **Input Output** | | | | | | | | |
| REGISTER INPUT | RIC | 00 | 100011 | | | | | |
| REGISTER OUTPUT | ROC | 00 | 100100 | | | | | |

This approach to the C and DX operand specification means that only a two-bit modification designation is required to define the six types of address modification, three bits to specify the index or operand register and possibly an additional word for address or immediate data

The design specified for the processor uses separate register designations (R7 and R0) for the PC and non-indexing functions, respectively This convention is maintained throughout the design specification There is, however, an alternative that is worth considering The use of the PC for indexing purposes permits program counter relative addressing, however, the merits of this mode are unclear with the use of ROM (write prevention) memories The use of the PC to distinguish non-indexing could essentially free up one more general register for use as an index base register for operand modifications

The next part of the instruction execution design to consider is the operation phase Three bits are required to specify the source destination register during the operation Two bits will be used as part of the extended short format designation. This assigns only 13 percent of the instruction bits to allow 40 percent to 55 percent of the instructions to use the one-word extended short format rather than a two-word standard format This leaves 6 bits to specify one of 64 possible standard format operations These 6 bits permit the specification of 64 unique operations, which should be more than enough for the DP M processor Actually only 30 basic standard format instructions have been defined for the processor

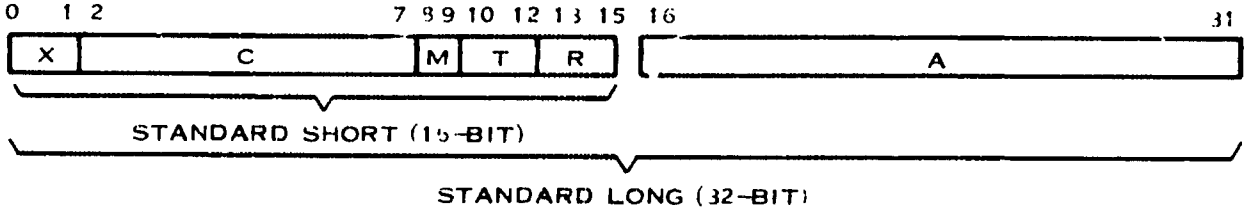
Figure 35 shows the standard instruction format and Table 7 lists the standard operations selected for the DP M instruction set

In Figure 35 the six fields of the standard instruction format defined for the DP M PE are

- X Two-bit field of all zeros for the standard format, used to differentiate between standard and extended short format instructions
- C Six-bit command field, used to specify the desired operation such as load, store, add, subtract, etc
- M Two-bit operand modification field, used in conjunction with the T field and possibly an A field to determine the operand [Derived Operand (DO) or Derived Address (DA)] to be used during the operations specified by the C field Figure 35 shows the various DOs and DAs as a function of the M, T, and A fields
- T Three-bit field which specifies the register or index base to be used as part of the operand Note that if register 0 or 7 is specified, special actions occur as described above
- R Three-bit field which specifies the accumulator to be used during the operation or the branch condition for a conditional branch.
- A The next instruction word in the instruction stream which if used contains a word of constant data or an address.

Note that the T and R field may be used to specify either a single register or a register pair

STANDARD FORMAT




X $00_2$ (FORMAT DESIGNATOR)
C OPERATION CODE
M MODIFICATION CODE (OPERAND DERIVATION MODE)
T TAG REGISTER
R OPERAND SOURCE/DESTINATION REGISTER, BRANCH CONDITION, OR BIT SELECTOR
A MEMORY ADDRESS IMMEDIATE DATA
(Y) REGISTER POINTED TO BY FIELD Y
[Z] MEMORY LOCATION POINTED TO BY Z



MODIFICATION

| M | T | DERIVED OPERAND/ADDRESS | FUNCTION |
|---|---|---|---|
| 00 | t | (t)/– | REGISTER |
| 01 | t | [(t)]/(t) | REGISTER INDIRECT |
| 10 | t (t≠7) | [(t)]/(t) AND (t) ← (t) + 1 | REGISTER INDIRECT AUTOINCREMENT (a) |
| | 7 | A/– | CONSTANT DATA (b) |
| 11 | 0 | [A]/A | DIRECT |
| | t (t≠0) | [(t) + A]/(t) + A | DIRECT INDEXED |

(a) USED TO SCAN THROUGH TABLES AND TO POP STACKS.
(b) INDIRECT AUTOINCREMENT USING THE PROGRAM COUNTER.

Figure 35. Standard Instruction Format

The 30 standard format instructions specified for the DP/M processor represent a minimum but adequate set of operations for expected real-time avionic environments. The capabilities of this instruction set include:

Memory bit field manipulation under a mask operation

Data movement instructions like Push and Move and Autoincrement for efficient list and table processing

Register file save/restore operations for interrupt processing and subroutine processing

A full complement of signed 16-bit arithmetic operations

A full complement of 16-bit logical operations

A full complement of bit manipulation operations that provide an uninterruptible memory read/modify/write capability

Single- and double-length shifts including arithmetic, logical and circular types whose shift counts are specified by the derived operand of the instruction

A conditional branch/transfer capability with full 64K address reach

Basic interrupt status save/restore instructions

Input/output instructions where the data command information to be processed is found in the register file.

The standard set of hardware instructions shown in Table 7 can be expanded by adding new operations (and more hardware) to use one of the 34 unused operation codes or by using an invalid "op-code" detect interrupt to simulate desired instructions with interpretive software routines.

### *b. Extended Short Format Instructions*

As mentioned earlier, a frequent use was identified for a few instructions that only need a short (8-bit) data or displacement field. The DP/M PE extended short instruction formats are shown in Figure 36. The usage of the DP/M PE extended short format instruction set is shown in Table 8. The extended short format duplicates the most common standard format operations but in a short format. The inclusion of this short format is based on the observation that a few

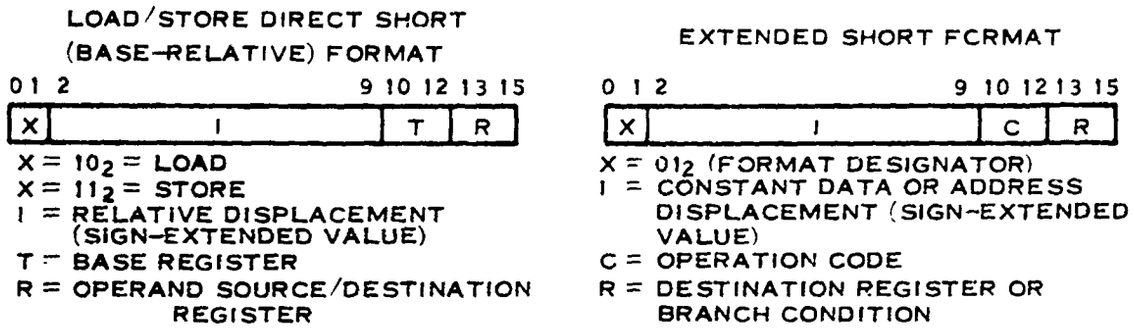



**Figure 36. Extended Short Instruction Formats**

and one for the subroutine entry address), once the linkage address is defined, all other subroutine calls require only a single word instruction.

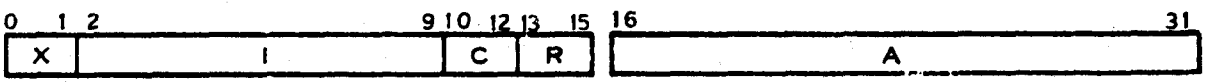
### c. *Input/Output Instructions*

The DP/M I/O operations use the standard format for register I/O and a variant of the extended short format for memory I/O. In both cases, there is an 8-bit I/O Command Address Word (CAW). This, in conjunction with the direction of data flow (in versus out), is used to specify which external device(s) are to respond and what action they are to take. In some cases, there is no data to be processed, but simply a command specified by the particular CAW. Thus, the 8-bit CAW and direction of I/O can access 512 devices (256 in and 256 out) or specify 512 operations or any combination. Normally, a CAW is directed at a particular device and the data word transmitted is the actual device command. This allows each of 256 devices to be given as many as 64K unique commands.

The register I/O instruction uses the derived operand as the CAW and inputs or outputs the data to or from the register specified by the R-field. The memory I/O instructions use a modified extended short format instruction (32-bits) (see Figure 37). The 8-bit immediate I-field is the CAW and the memory location is specified by the second half (16 bits) of the instruction. If the R-field is other than register 0, indexing takes place. To keep from "hanging" the processor waiting for an external device to acknowledge an I/O operation, the I/O control chip performs any required timeout function necessary to allow a device data receipt/acknowledge sequence to be generated. If the timeout on I/O occurs, the processor-condition code will be set to reflect the failure of the I/O operation. Further, if the I/O timeout interrupt is enabled (device error mask), an I/O interrupt trap will also occur.

### d. *Summary of Processor Instruction Set Characteristics*

To reduce the complexity of the instruction decode, the number of formats has been limited and the instruction cycle standardized. The majority of the instructions are in the standard format that allows for a large number of operations (64) and a full range of addressing modes (6). Added to this powerful set of instructions is a small set of extended short format instructions. These short format instructions implement the more frequently executed instructions. These short format instructions can save 25 to 35 percent of the space and




$X = 01_2$
I = COMMAND ADDRESS WORD (CAW)
C = OPERATION CODE
R = INDEX REGISTER (NO INDEXING IF R = 0)
A = MEMORY ADDRESS

Figure 37. Memory Input/Output (Special Extended Short) Format

## TABLE 9. DP/M INSTRUCTION USAGE ANALYSIS SUMMARY

| Instruction or Modification | Sample* Static Usage (Percent) | DP/M Executive Static Usage | | Version 1.0 Diagnostics | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | Static Usage | | Dynamic Usage | | |
| | | (Count) | (Percent) | (Count) | (Percent) | (Count) | (Occurrence Percent) | (Execution Time Percent) |
| LCS | 5.0 | 25 | 5.6 | 36 | 6.3 | 35 | 5.8 | 3.4 |
| ACS | 2.0 | 27 | 6.1 | 12 | 2.1 | 36 | 6.0 | 3.5 |
| BS** | 5.0 | 11 | 2.5 | 14 | 2.4 | | | |
| CCS | 4.0 | 5 | 1.1 | 39 | 6.8 | 42 | 7.0 | 4.1 |
| BCS | 13.0 | 61 | 13.8 | 209 | 36.4 | 208 | 34.7 | 20.4 |
| BILR | 3.0 | 12 | 2.7 | 1 | 0.2 | 1 | 0.2 | 0.2 |
| IBNS | 0.5 | 19 | 4.3 | 1 | 0.2 | 10 | 1.7 | 1.9 |
| LDS | 5.0 | 50 | 11.3 | 3 | 0.5 | 3 | 0.5 | 0.6 |
| STDS | 2.5 | 33 | 7.5 | 1 | 0.2 | 1 | 0.2 | 0.2 |
| Extended Short | 40.0 | 243 | 54.9 | 316 | 55.1 | 336 | 56.1 | 34.2 |
| R, RI, RIA | 10.0 | 91 | 20.5 | 48 | 8.4 | 45 | 7.5 | 12.7 |
| All Short (16-Bit) | 50.0 | 334 | 75.4 | 364 | 63.5 | 381 | 63.6 | 46.9 |
| C, D, DX (Long 32-Bit) | 50.0 | 109 | 24.6 | 210 | 36.5 | 218 | 36.4 | 53.1 |
| Total | 100.0 | 443 | 100.0 | 574 | 100.0 | 599 | 100.0 | 100.0 |
| Extended Short Format Savings | | | | | | | | |
| Memory | 30 percent | | 35 percent | | 35 percent | | | |
| Execution Time | - | | - | | - | | | 25 percent |

*Based on a 12,000 instruction EW program with an instruction set similar to the standard format.
**BS is executed as an ACS to Program Counter.

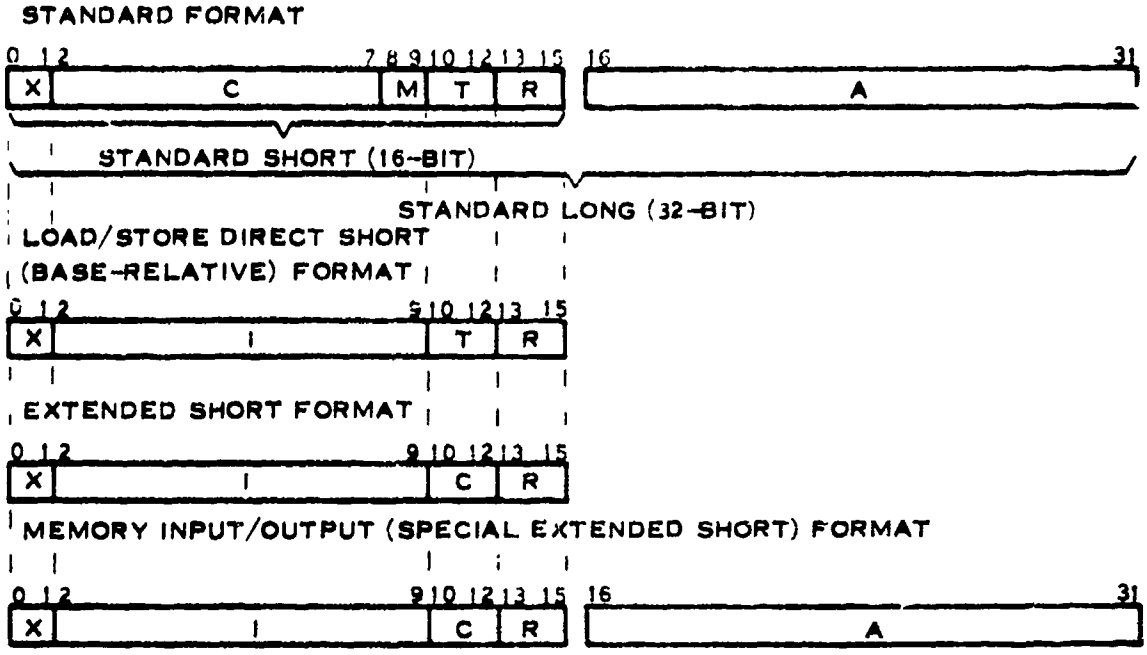



**Figure 38. DP/M Processor Instruction Formats**

execution time of a program. Figure 38 shows how the fields of the standard and extended short format instructions are aligned. This reduces the amount of hardware logic required to implement instruction decoding. Tables 7 and 8 list the processor instruction set. Any of the invalid instructions can be used to implement a trap and emulate operation. This would allow the inclusion of extra operators through the inclusion of the appropriate software.

## 5. Instruction Usage Observation

Several different programs and sample cases were studied during the course of the DP/M processor design specification. The first such operational software module was a 12,000 instruction Electronic Warfare (EW) program. This program was written for an avionic digital computer which had an instruction set similar to the DP/M standard format instructions. The next programs examined were the DP/M executive and one version of the DP/M instruction diagnostic program. The results from these three programs are summarized in Table 9. The use of register indirect autoincrement for manipulating data structures is shown and the applying of the DP/M instruction set to a simple FORTRAN subroutine linkage with "pass by reference" variables is presented.

### *a. Extended Short Instruction Format Usage Analysis*

The sample static instruction usage shown in Table 9 is based on postulating the effect of the extended short instruction format on the aforementioned EW program written for a digital computer whose instruction set was similar to the DP/M standard instruction format. The

analysis was performed by running the operational flight program source code through a program which counted the number of times that each operation/modification combination occurred. The extended short format instruction usage was then estimated by counting the standard format operation/modification pair that corresponded to the extended short instructions. Then, the fraction of that usage that could have used a short (8-bit) constant value rather than the long (16 bit) constant value was estimated. This instruction usage data corresponds to the static case only. Clearly, the highest usage was for short relative branching, both conditional and unconditional, which accounted for almost 20 percent of the instructions used. The increment and branch if negative instruction has been included because of its relatively high dynamic usage for loop processing. Its function is to increment the register specified and to branch if the register is negative. The subroutine call instruction saves the current program counter (PC) in the register specified and branches indirectly through one of the first 256 memory locations. This instruction makes subroutine linkage efficient and supports modular programming. There is also a high usage of short immediate data for count and address processing. Count processing records number of events, loops, etc. Thus, the most frequently used instructions for short immediate data (−128 to +127) are load, add, and compare. These three operations amount to more than 10 percent of the static instruction usage. The next most frequently occurring class of instructions is loads and stores that are PC or base-relative with an address displacement from the base of −128 to +127. This sample case shows a 30-percent savings in memory space due to these eight extended short instructions formats.

Portions of the code written for the DP/M processor instruction diagnostic and executive were also analyzed. The results were somewhat better overall than that estimated from the EW program. This is probably due in part to a conscientious effort to use the extended short instructions since they were available. Both the executive and the version 1.0 instruction diagnostics showed a 30-percent program memory savings by using the 16-bit extended short instructions rather than the 32-bit standard format instructions (e.g., BCS - 16 bit versus BC - 32 bit). The dynamic (simulation) data was also available for the version 1.0 instruction diagnostics. The static usage was very close to the dynamic usage except for the IBNS which was used for loop closing. The extended short and the corresponding standard format instructions have execution times limited by the memory cycle speed. Thus, there is nearly a 2 to 1 savings in execution time of an extended short instruction over its standard format equivalent. In the case of the version 1.0 diagnostics, a 25-percent savings in execution time resulted from using the extended short instructions.

**TABLE 10. DOUBLE PRECISION OPERATION USAGE**

| | | Percent |
|---|---|---|
| Load | | |
| | Constant | 0.6 |
| | Direct | 1.0 |
| | Direct Indexed | 0.1 |
| | Total | 1.7 |
| Store | | |
| | Direct | 0.9 |
| | Direct Indexed | 0.4 |
| | Total | 1.3 |
| Add | | |
| | Register | 0.2 |
| | Constant | 0.1 |
| | Direct | 0.1 |
| | Total | 0.4 |

*b. Effect of Instructions Not Included*

(1) Double Precision

The EW program had a small usage of double precision operations, which are shown in Table 10.

Clearly, the 3-percent double load/store operations will not add significantly to either the program space and/or execution times since they can easily be accomplished by simply using two single load/store operations. There were no double arithmetic operations except for the 0.4-percent double adds. Thus, even if

they took 10 times as long to simulate, they still would not greatly affect program execution time.

(2) Bytes

The EW program, as well as several other much larger avionic programs which were examined, did extensive display manipulation, but in none of the cases was the lack of character (byte) manipulation considered a serious problem. Even though there was extensive display processing, it did not constitute a large processing load. Thus, there was plenty of time to manipulate character data by using shifting and masking (AND) operations.

(3) Floating Point

The EW problem and the several others examined could have used floating-point data and operations. The increased hardware cost and loss of processor performance, though, could not be justified and the next generation 16-bit commercial microprocessor is also not likely to have these operations.

Based on this information, it is difficult to justify the additional hardware to implement these added operations. If some operations were to be added, the following ranking seems appropriate:

Double precision add

Double precision load/store

Character (byte) manipulation

### *c. Data Structures Using Register Indirect Autoincrement*

The register indirect with autoincrement operand modification, in addition to being used for constant data and scanning through tables, can be used for more sophisticated data structures such as queues and stacks. A queue can be implemented by using two registers as pointers to the queue. One pointer is for the input and the other is for the output. The pointers are then scanned through the queue by using the register indirect with autoincrement modification on the instructions that access the queue. Similarly, a stack can be defined using one register whose contents serve as a pointer to the top of the stack. One way of defining a stack (last in-first out list) is to define the top to have a smaller address than the bottom (Figure 39). This allows the use of register indirect with autoincrement to remove (POP) data off of the top of the stack. With this definition, all that is required is a single "push" instruction to store data into a stack. The push instruction is used to store data into a stack just like a store instruction would be used for other types of data structures. This type of stack structure is used for interrupts where register 6 serves as the interrupt stack pointer. This interrupt stack can also be used for register file savings and restoring for interrupts and subroutines via the push and pop multiple instructions.

### *d. Sample Subroutine Linkage*

While at first glance, it may have seemed that, due to the lack of the indirect addressing, there would be significant inefficiencies in subroutine parameter passing. This is not true with the DP/M PI. instruction set. An instruction normally would take two words if indirect

addressing were available (one word of operation, addressing mode and register and one word of address/displacement). Without indirect addressing, though, it can be done on the DP/M by using an extended short format load direct short instruction to get first the address and then a standard format instruction using register indirect. This sequence uses the same number of memory locations and operates at about the same speed as would the two-word indirect addressing instruction.

As an example, consider the following FORTRAN "pass-by-reference" operation (Figure 40) where the addresses of the calling parameters are placed in the instruction stream following the branch and link to subroutine. Assume the following register usage:

| Register | Use |
|---|---|
| 0 to 3 | Computation |
| 4 | Indirect Addressing |
| 5 | Return Link |
| 6 | Interrupt Stack Pointer |
| 7 | Program Counter |

And where,

Standard Format

L–Load

ST–Store

A–Add

R–Register Modification

RI–Register Indirect Modification

Extended Short Format

LDS–Load Direct Short

ACS–Add Constant Short

BILR–Branch Indirect and Link Register

Assembler Directive

DC–Declare Constant (one word of address or label).

This code sequence takes the same program space and executes in nearly the same time as the more conventional type of indirect addressing were used. There was no attempt to optimize the above code by using autoincrementing to step through the parameters or other such techniques. The above code is representative of that which a simple compiler would generate. It does, however, show the flexibility of the DP/M instruction set and its ability to support higher-order language-linking conventions. It should also be noted that, when passing an array address by reference, both pre- and post-indexing are required. This can be complicated with conventional instruction sets but it can be handled easily with the DP/M baseline PE instruction set.

From the existing and new programs that were examined, the DP/M instruction set seems to be very flexible and efficient. It would be interesting to see if the previous programs, such as

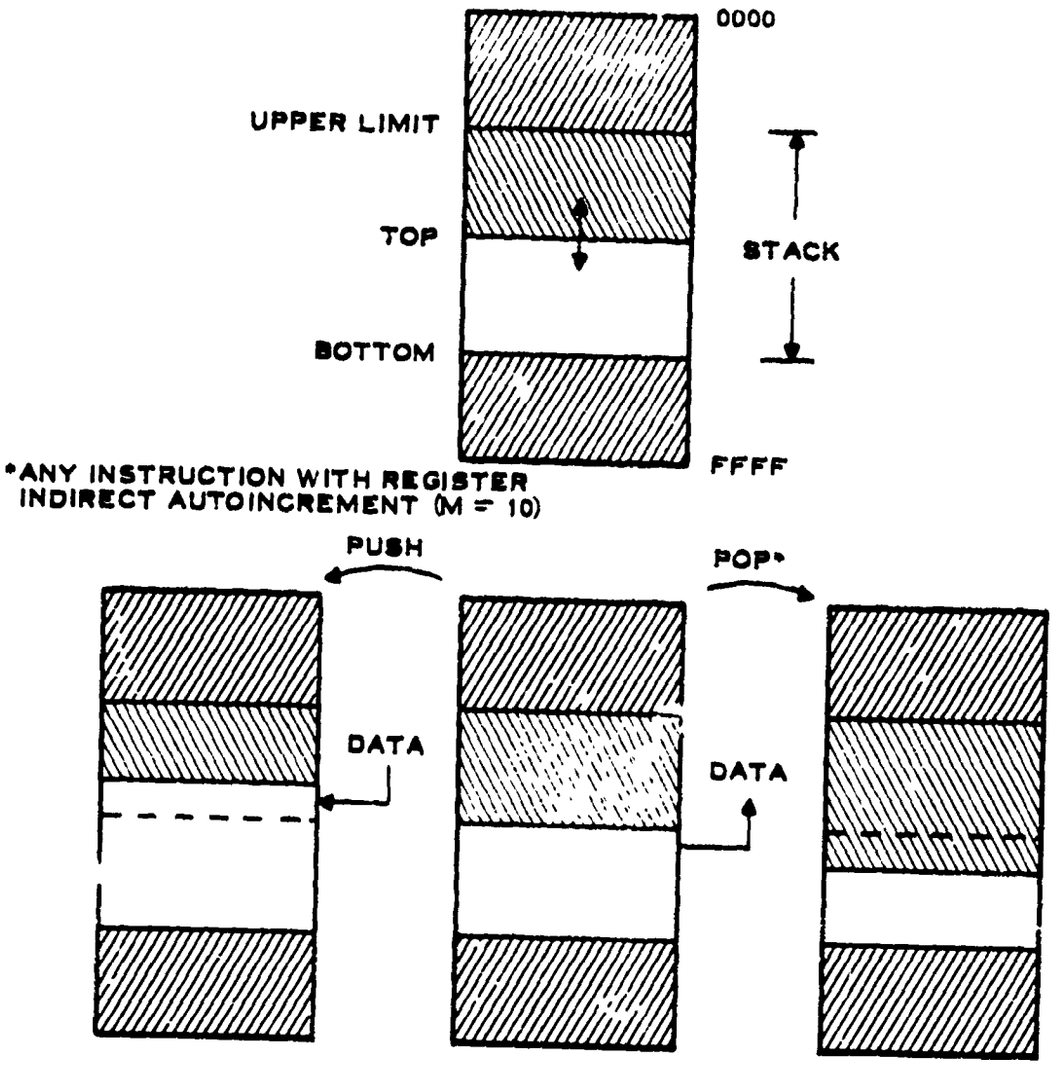



Figure 39. DP/M Stack Operations

the EW one, would show the same sort of improvement that the DP/M executive and diagnostic program did. That is, would the programmers find more ways of making use of the extended short format and register-indirect/register-indirect autoincrement modifications if these options had been available.

## 6. Processor Interrupt Structure

This subsection covers the interrupt structure as viewed by the processor. An attempt has been made to minimize the required hardware in the processor while providing as much system capability as possible. The interrupt structure can be broken down into two functional parts. The first is the controlling or enabling of individual interrupts and the other is the switching of the machine state (context switching). This interrupt control can be accomplished by having one or

FORTRAN

```
      SUBROUTINE TWOSUM (IN1, IN2, IOUT)
      IOUT = IN1 + IN2
      END
      MAIN
           •
           •
           •
100   CALL TWOSUM (IA, IB, IC)
200        •
           •
           •
      END
```

DP M MACHINE CODE

```
     *SUBROUTINE TWOSUM (IN1, IN2, IOUT)
     *IOUT = IN1 + IN2
      LDS    R4, R5, 0        GET ADDRESS 1ST PARAMETER
      L.RI   R0, R4           LOAD 1ST PARAMETER
      LDS    R4, R5, 1        GET ADDRESS 2ND PARAMETER
      A.RI   R0, R4           ADD 2ND PARAMETER
      LDS    R4, R5, 2        GET ADDRESS 3RD PARAMETER
      ST.RI  R0, R4           STORE 3RD PARAMETER
      ACS    R5, -3           STEP OVER PARAMETERS
      L.R    R7, R5           RETURN
     *END
     *MAIN
           •
           •
           •
     *CALL TWOSUM
100   BILR R5, TWOSUM         CALL TWOSUM
      DC   A                  }
      DC   B                  } ADDRESSES OF PARAMETERS
      DC   C                  }
                              RETURN POINT
200        •
           •
           •
     *END
```

Figure 40 Sample FORTRAN Subroutine Linkage

more levels of interrupt masks. For example, there can be a primary interrupt mask with each bit in it controlling whether or not any interrupt will be allowed from a particular level of interrupts. In this case, there could be multiple secondary-level masks that enable individual interrupts, which are at the same level. The second functional part of an interrupt sequence is the initiating of the desired action based on which interrupt occurred. This can be done with either hardware or software. When it is done with hardware, usually an interrupt trap vector is used. The interrupt trap vector memory locations contain the address of the interrupt service routine and the new processor status word(s). The hardware then saves the old program counter and status word(s) and loads the new ones from the interrupt trap vector. An alternative implementation is to have the interrupt trap locations contain an instruction which is executed out of sequence. The latter approach is much more efficient in the case where the interrupt routine is only one instruction long (e.g., counting) but is less efficient for entering longer routines. If there is not a hardware trap vector, the same type of action can be implemented in software.

### *a. Interrupt Control Structure*

The first item to be decided concerning the interrupt control structure is whether it should be single-level or multiple-level and, if multiple-level, how many levels should be provided. The single level requires the least hardware but at the expense of slow interrupt response and increased processor software. By carefully determining which functions must be provided by the processor and which can be external to the processor, it was concluded that the hardware required in the processor could be quite minimal. Thus, by providing minimal hardware in the processor, the interrupt structure can be as sophisticated or simplistic as system requirements dictate and/or external hardware provides.

Thus, for simplicity and flexibility, the interrupt structure of the DP/M PE is distributed between the processor, I/O, and BIU. Each portion controls that part of the interrupt structure that is appropriate to it. As such, the processor controls its own internal interrupts (overflow, invalid op-code, I/O, and memory timeout) and all interrupt masks. The processor has a status word (Figure 41) which contains a two-bit condition-code register, an overflow indicator and interrupt mask, a device-error interrupt mask, a memory-error interrupt mask, and 10 bits of mask for interrupts external to the processor (i.e., the I/O and BIU). These latter 10 bits are external to the processor chip and reside physically on the I/O and/or BIU chips, and are accessed by the processor performing an I/O operation with a CAW of 00 (Hex) as a part of any

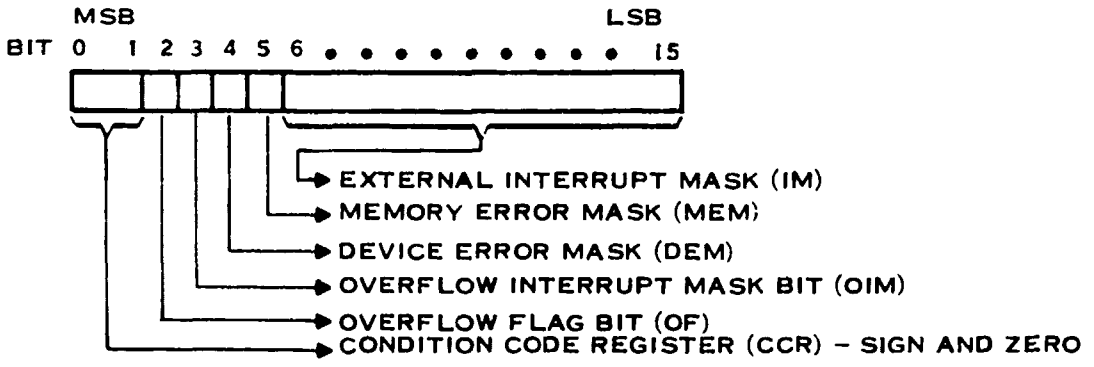



Figure 41. Status Word Format

status word read/write instruction. Thus, whenever a status word read or write instruction is executed, the processor directly accesses its own 6 bits, and accesses (to the processor) the remaining external 10 bits by simulating an I/O instruction with a CAW of 00 (Hex). These 10 external interrupt mask bits can be used to directly control interrupts or levels of interrupts. In the case of an interrupt level, the status word bit would enable or disable a whole interrupt level. Then there would be additional external mask word(s) which would be used to control the enabling and/or disabling or individual interrupts. These additional external mask word(s) would be accessed via other I/O instructions.

### *b. Interrupt Context Switching*

As with the interrupt mask, the interrupt-initiated response is distributed between the processor, I/O, and BIU. The processor supports internally and externally vectored interrupts. Thus, when an interrupt occurs, the current program counter and status word are saved, and the new values are loaded from a pair of memory trap locations unique to the particular interrupt. In the case of internally generated interrupts, the processor determines the trap locations. For all external (I/O and BIU) interrupts, the trap locations are specified by the interrupting unit in response to the interrupt acknowledge.

As mentioned above in servicing an interrupt, the current program counter and status word must be saved so that the interrupted program may be resumed. The question then is where should these words be saved. There are several candidate approaches including:

Dedicated Registers In this case there is a dedicated set of registers that hold the previous program counter and status word. This works well for the first interrupt, but handling of subsequent interrupts requires saving these registers somewhere else. Thus, the basic problem has not been solved, only forestalled. Another disadvantage is the additional registers and hardware required in the processor.

Dedicated Hardware Stack In this case, the PC and status word are pushed into a hardware stack. This solves the problem of what to do with subsequent interrupts but at the expense of even more hardware. It also has a limited nesting of interrupts, limited by the size of the hardware stack.

Single Dedicated Memory Location In this case, the PC and status word are saved into a fixed pair of memory locations. This has the advantage of minimum hardware. It still suffers as "a" above; namely, what to do with subsequent interrupts.

Separate Dedicated Memory Locations In this case, the PC and status word are saved into different pairs of memory locations for each interrupt level. This requires little hardware and solves the case of multiple interrupts as long as no more than one interrupt is allowed per level. This does present a serious problem, though, when the memory size and type is variable (ROMs and RAMs). That is, the trap vector wants to be in ROM and the save vector must be in RAM.

Memory Stack In this case, the PC and status word are pushed into a stack pointed to by a hardware register. This allows virtually unlimited nesting of interrupts from a different or the same interrupt level. It also allows arbitrary separation between the interrupt trap vector and the save area.

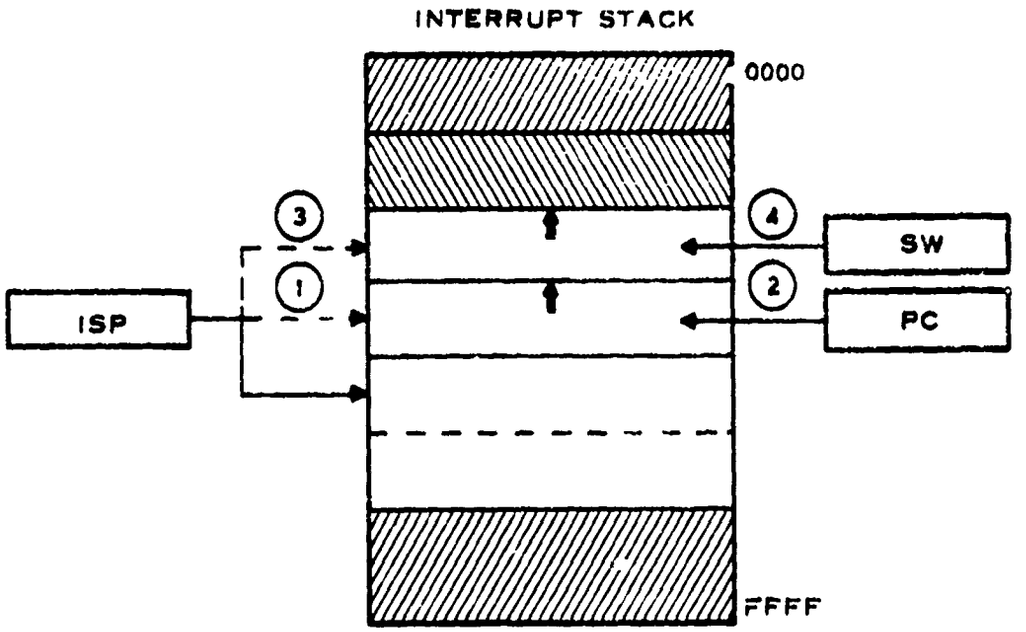



**Figure 42. Restoring of Program Counter and Status Word**

This latter approach is the one chosen for the DP/M processor because it appears to offer the greatest flexibility with a minimum of extra hardware. Figure 42 shows the status word (SW) and the program counter being stacked (PUSHed) onto the interrupt stack using the interrupt stack pointer (ISP). Figure 43 shows the SW and PC being restored (POPed) from the interrupt stack. The next question is whether the stack pointer register should be an additional register or one of the general registers in the file. The advantages of using one of the general registers are:

Less hardware

No extra instructions needed

Can be used as part of a general memory management scheme for re-entrant coding.

As previously noted in the register discussion, the disadvantage is that this approach uses an additional general register. But, if there is any re-entrant programming, a stack and a stack pointer are required. Thus, for the DP/M PE, general register 6 is dedicated to being the interrupt stack pointer.

In summary, the DP/M processor supports multiple-level interrupt masking, vectored interrupt trapping, and a memory stack for saving the machine state (PC and status word).

## 7. Processor Initialization

The initialization of the DP/M PE consists of setting the program counter of the processor to the beginning of the initialization routine that is usually a bootstrap loader program in read

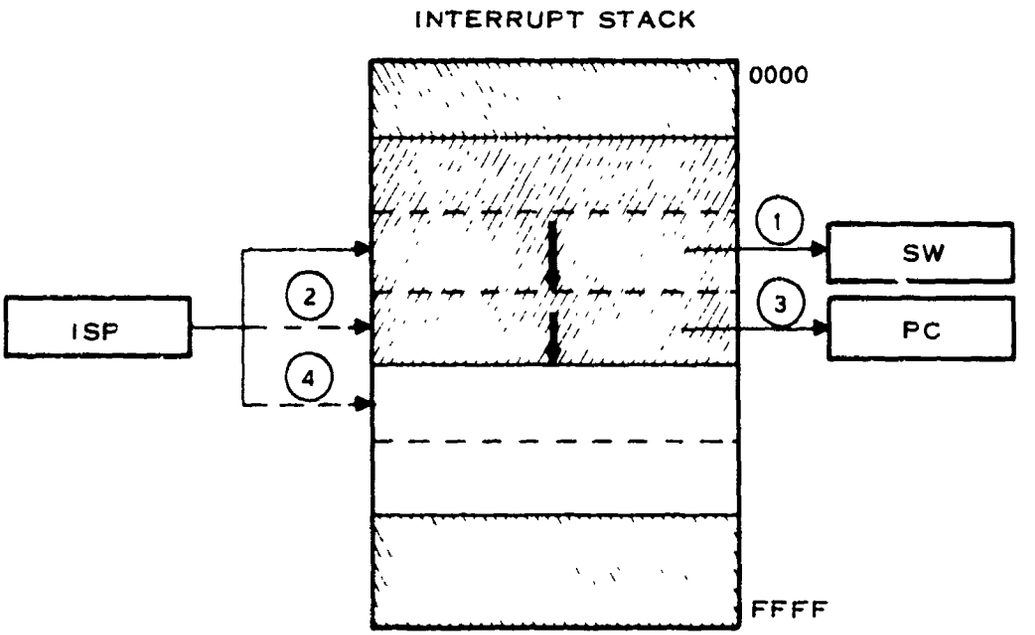



**Figure 43. Restoring of Program Counter and Status Word**

only memory (ROM) which then loads a program module. This program module can be the total program or another more powerful loader which then loads the desired program(s).

The initialization start address is sent to the processor over the internal processor bus (Subsection IV.D) similar to the way that the interrupt trap address is sent to the processor. That is, the address is placed on the internal data lines and the proper control lines are activated. This signal forces the program counter to be loaded from the I-BUS data lines and instruction fetch to be initiated. Therefore, it is the responsiblity of the element which issues a CLEAR/RESET signal to supply the processor with its start address. This approach allows the most flexibility for the processor while minimizing the necessary hardware.

## 8. Processor Functional Design Summary

This functional design offers a powerful but realizable microprocessor for the avionics application. Functionally, the DP'M processor is a 16-bit, two's complement, 8-general-register machine. These 8 general file registers are used as:

1–Program Counter (register 7)

1–Interrupt and Temporary Stack Pointer (register 6)

6–General Purpose Accumulators (registers 0 through 5)

7–Standard Format Index/Base Registers (registers 1 through 7)*

*Or, if the direct indication is changed from 0 to 7, then registers 0 through 7.

8 Load/Store Direct Short Base Registers (registers 0 through 7)

The primary data types are single bits and full 16-bit words. Neither double precision (two-word) arithmetic nor byte (character) processing has been included due to their expected low usage. Floating-point operations have been omitted also because the additional hardware would extend the processor beyond what would be considered reasonable for a commercially available microprocessor for the 1978 80 time period. The primary candidates for instruction set expansion would be a double precision add followed by double word load and store operations. Of less importance would be double precision subtract and byte manipulation. The DP/M base line instruction formats are summarized in Figure 38 and the operations are summarized in Tables 7 and 8. The DP/M processor has a full complement of operations composed of 40 basic instructions of the following types:

Arithmetic

Logical (Boolean)

Data Transfer

Shifts (logical and arithmetic)

Bit Manipulation

Compare

Program Transfer (conditional and unconditional)

Subroutine Linkage

Control (status word)

Interrupt (Context Switching via stack)

Input/Output

The DP/M instruction set reflects the special emphasis that is placed on several design areas:

Maximization of instruction code efficiency both execution time and memory usage (extended short instructions)

- Basic read/modify/write memory operations (BIT and STTM)
- Ease of access to small local workspace (register file and LDS STS)
- Loop control and indexing (RIA and IBNS)
- Subroutine linkage (BILR)
- Interrupt handling (multilevel with trap vector

No self-modifying code required (e.g., instructions whose characteristics are determined at run time) an important feature in read-only memory applications.

Flexible, efficient addressing mode (R, RI, RIA, C, D, DX)

As was observed in the programs analyzed, the extended short format instructions were used between 40 and 55 percent of the time which contributed to a 30 to 35 percent savings of program memory and 25 percent savings in program execution time. The instruction set has been selected to make subroutine linkage easy and to support the basic types of data structures such

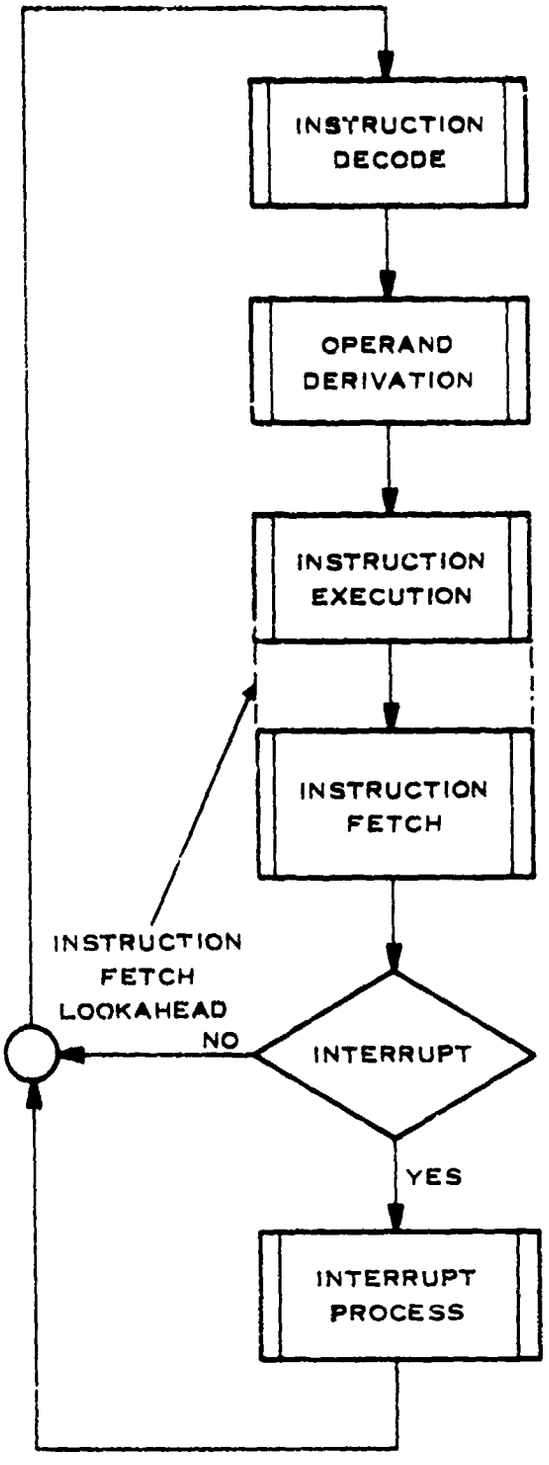



Figure 44. Basic Instruction Cycle With Interrupts

as lists, stacks, and queues. The interrupt structure uses an interrupt stack to save the program state and to allow multiple nesting of interrupts.

The intent of the design effort was that functionally DP/M base line processor be a powerful and flexible machine. At the same time, special consideration has been given to seeing that the processor is not too complex to be implemented as one to two microprocessor chips.

## 8. PROCESSOR PHYSICAL DESIGN

The processor physical design is characterized by the actual hardware registers and memories, arithmetic and logic units, and the data and control paths. There may or may not be a close correspondence between the actual hardware architecture and the functional architecture that the programmer sees. As previously cited, the register file may actually be part of the main memory rather than being an actual hardware register file. Thus, the actual hardware may be less than the programmer thinks it is. On the other hand, there are frequently a number of internal working registers that hold intermediate results that are never seen by the programmer. Figure 45 shows the overall general block diagram of the DP/M PE processor. It is composed of the instruction register and decode section, processor control section, micro data processor, and status register. Two primary considerations are the construction of the processor control (hard-wired versus micro-controlled) and whether the general register file should be in the micro data processor or in main memory. Based on current trends in processor design, the processor control will use a combination of micro code and programmable logic arrays (PLAs). PLAs are especially useful for control applications in semiconductor devices (combinatorial logic) where it is a requirement to alter (change or correct) the control without completely redesigning the chip. This flexibility

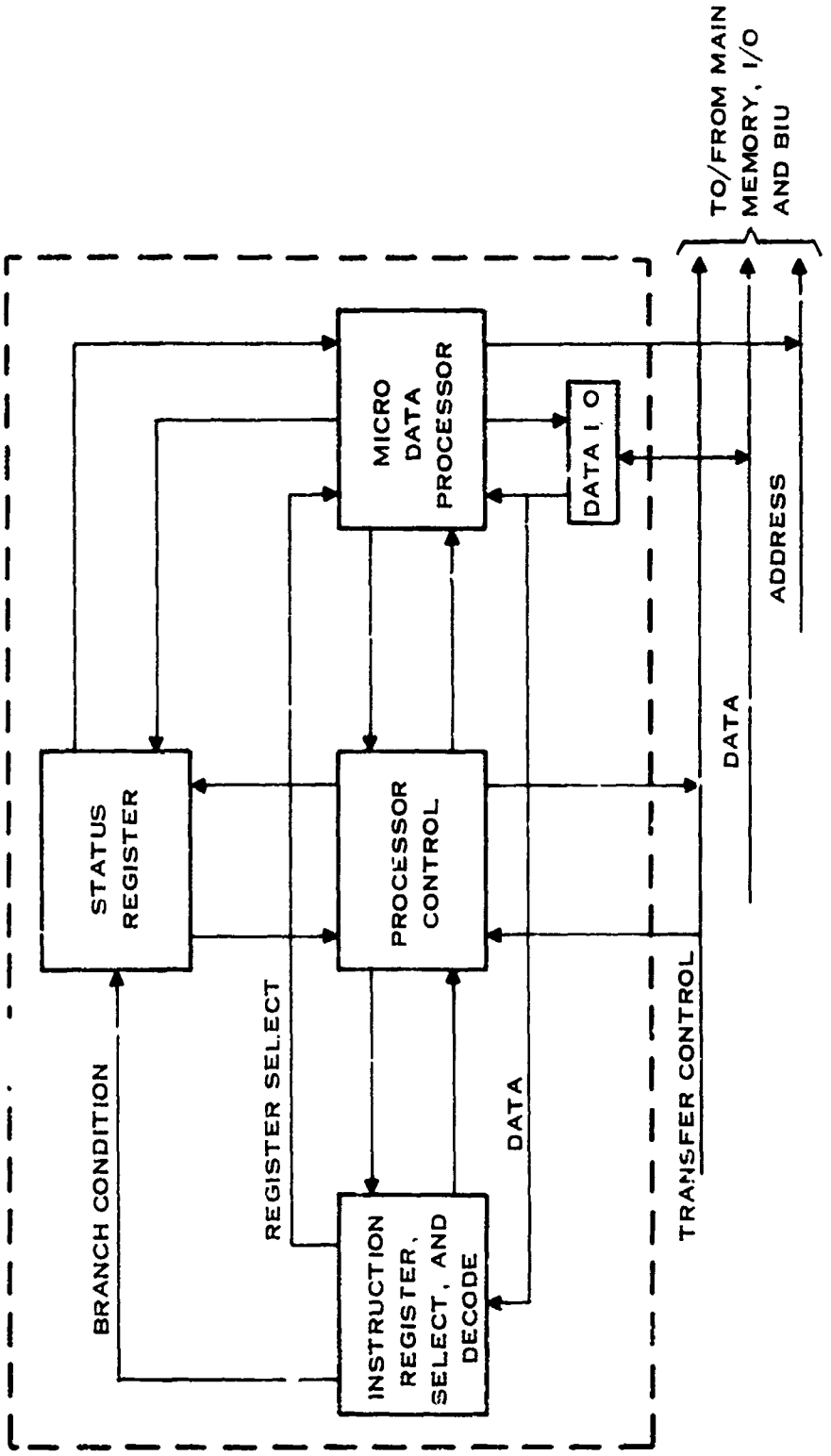



Figure 45. Processor Block Diagram

is enhanced by the high device packing density available with PLAs. The recommended approach for the DP/M processor is the use of a combination of the following control techniques:

(1) hardwired data paths, registers, and ALU

(2) PLAs for control decode at the clock level

(3) Read Only Memory (ROM) for clock-to-clock sequential control.

Furthermore, it is recommended the micro-data processor have a hardware register file instead of a virtual (memory) register file. An 8-word by 16-bit hardware register file increased the overall processor gate count by slightly more than 10 percent and offers a potential two to one increase in performance. The following paragraphs discuss the PE instruction execution cycle, the hardware block diagram of the processor module, and develop a gate level complexity estimate for the processor.

## 1. PE Instruction Cycle

The DP/M PE instruction cycle can be divided into those phases shown in Figure 46. Some of these operations may actually be overlapped in time; however, it is convenient to view them as sequentially executed. The four major operations in the instruction cycle are.

*Instruction decode* which consists of determining what is the first action that the new instruction requires. This first action will be the derivation of operands for use during the instruction execution cycle.

*Operand derivation* which is the generation or fetching of an address or data to be used during operation execution.

*Operation execution* which uses the address or data supplied by the operand derivation and carries out the desired operation specified by the instruction.

*Instruction fetch* which consists of loading the instruction register with the memory word that the program counter is pointing at and incrementing the program counter.

### *a. Instruction Fetch Lookahead*

The simplest form of pipelining or lookahead is instruction fetch lookahead. This can be accomplished by combining the execution of the current instruction with the fetching of the next instruction. This lookahead is possible as long as the current execution does not involve main memory (e.g., a store instruction) or the program counter (branch instruction). In theory, this could result in a two to one improvement in the execution time for single word instructions (extended short and register-to-register operations that are not branch or store instructions). This can amount to approximately 25 percent of the typical mix of avionic program instructions. In practice, due to instruction mix and execution overhead, the improvement is limited to about a 10- to 20-percent improvement.

### *b. Instruction Execution Time With Instruction Fetch Lookahead*

The execution time of an instruction is the length of time to go completely through the instruction execution cycle. The instruction fetch time, without lookahead, is independent of the

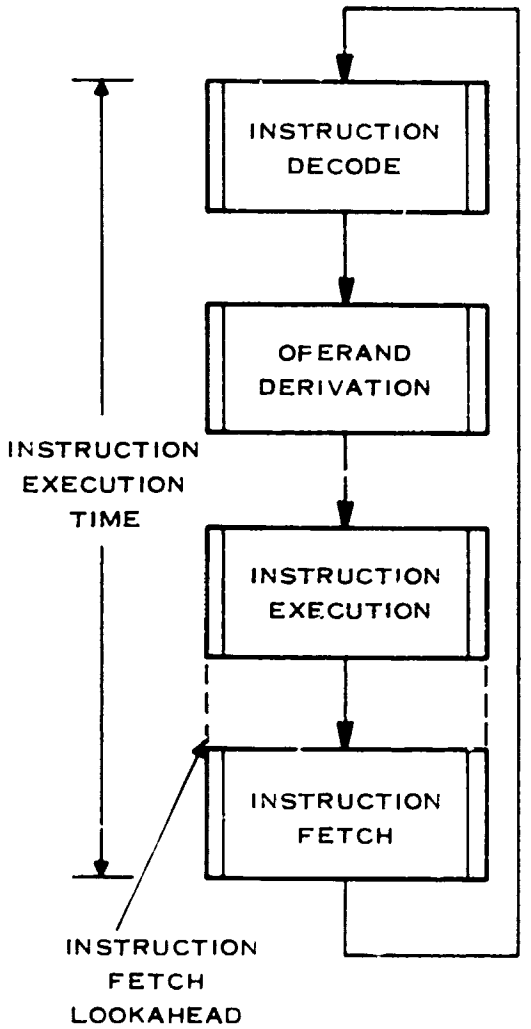



**Figure 46. Basic Instruction Cycle**

previous or the next instruction. Thus it does not matter whether we measure between the start of instruction decode or between the start of instruction fetch. With the effect of instruction fetch lookahead on such instructions as branch and store, the instruction fetch time is dependent on the preceding instruction. In this case, it is best to define instruction execution time as the time between starts of instruction decoding. This makes the execution time of an instruction independent of the previous and the next instruction, even with instruction fetch lookahead.

### c *Interrupts With Instruction Fetch Lookahead*

Proper handling of internally generated (within the processor module) interrupts (i.e., overflow) can present special problems with instruction fetch lookahead. This problem results when the next instruction has already been fetched and the program counter incremented before any execution dependent interrupt can be detected. The best solution to this problem is to go ahead and fetch the next instruction and increment the program counter while the current instruction completes execution. Then, during instruction decode and just before the operand derivation starts for the next instruction, a test is made for pending interrupts. If there is an interrupt pending, execution of the new instruction just fetched is aborted, the program counter is restored to its previous value (i.e., it is decremented), and the appropriate interrupt service sequence is performed (Figure 44).

Arithmetic overflow is the primary internal (to the processor) interrupt. As such, it deserves special attention in the design of the processor. There are three overflow situations of interest to the programmer. The simplest is where overflows are to be ignored such as during modulo number processing of angles. The next situation is an overflow during a series of arithmetic calculations. Since this case can be an abnormal situation, it is important that its processing overhead be minimized. The normal way of processing these two cases is to either provide a maskable overflow interrupt and or to save the overflow status for subsequent testing. The last situation of interest is an overflow during the execution of a particular instruction where immediate corrective action must be taken. An example is software multiple precision routines. This case is best handled by following the instruction that may cause an overflow with an overflow test instruction (e.g., branch on overflow). In this case, an overflow interrupt should

be suppressed, since it is being tested explicitly. Because the next instruction has already been fetched before testing for an interrupt, it is a simple matter to test for an overflow followed by a branch-on-overflow instruction. In the DP/M Processing Element, the overflow followed by the branch-on-overflow instruction is tested before the test for interrupts (Figure 47). The branch-on-overflow instruction will clear the overflow bit. Thus, it is not necessary to change the overflow interrupt mask if there is to be immediate explicit handling of overflow. In addition to the intraprocessor overflow interrupt condition affecting the instruction execution cycle, a similar event can occur with an I/O instruction that causes either an interrupt to be generated (I/O time out or acknowledge) or requires only a condition code setting response.

### *d. Instruction Microcode Sequence*

The previously described DP/M instruction execution cycle shown in Figure 47 was extended during the design effort to include sample microcode sequences for different PE instructions. A brief review of the microcode control sequence defined for the PE is presented in this subsection. This microcode sequence applies to the physical hardware of the processor defined in Subsection IV.B.2.

The DP/M micro control sequences philosophy anticipates the more frequently used PE instructions and provides lookahead features to minimize the execution time for these operations. The micro control sequence starts when a new instruction has been fetched and placed in the instruction register and the sign-extended I-field of the instruction has been loaded into an internal working register, the $\mu$MQ. The instruction decode then generates an entry point to the beginning of the appropriate microcode sequence. This entry point is also used in response to an interrupt or to correspond to the beginning of a load/store direct short, extended short, operand/address derivation, or a register-to-register standard format instruction execution.

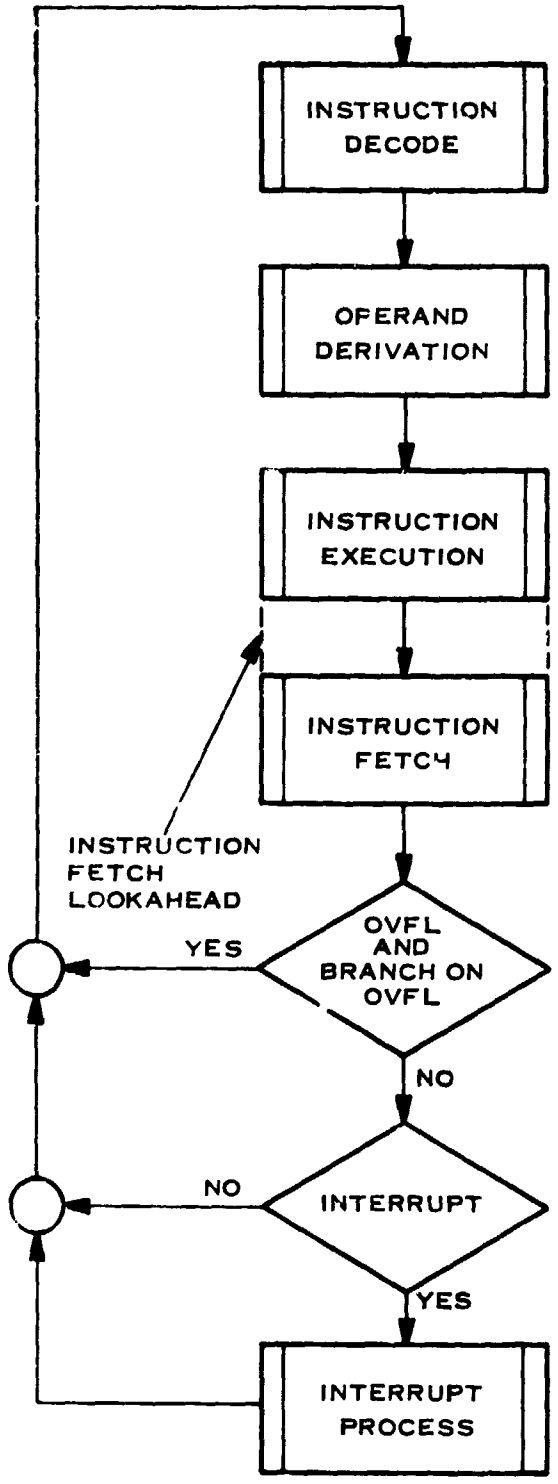



**Figure 47. DP/M Instruction Cycle**

Figure 48 shows the microcode sequence for a DP M direct indexed operand derivation cycle. Remember, the instruction fetch cycle is part of the final operation of the preceding instruction, and note that, during the instruction decode the register pointed to by the T-field of the instruction is loaded into the micro accumulator ($\mu$AC). This action allows the bypassing of the operand derivation cycle for register-to-register instructions, thus saving several micro steps and reducing instruction execution time.

Figure 49 shows the execution portion of a LOAD instruction, while Figure 50 shows the execution portion of an ADD instruction. Note that during the operand derivation cycle, the operand was placed in the $\mu$AC and the instruction execution microcode entry point was generated. For these simple instruction operations, only a single micro step is required. Also, because of independent program counter and address selection output control, the fetching of the next instruction is overlapped with the execution of the current instruction.

As previously noted the $\mu$MQ register will have been loaded with the sign extended I-field of the instruction during instruction fetch. Also, at the end of the decode cycle, the $\mu$AC will be loaded with the register pointed to by the T-field of the instruction. Because of this, the Load Store Direct Short, extended short, and standard format register-to-register operations can be executed immediately after decode.

Figures 51, 52, and 53 are examples of the micro sequences for the following instructions:

Load Direct Short

Load Constant Short

Load Register.

A comparison of Figure 49 with Figure 53 reveals that the standard Load instruction sequence is independent of the operand derivation.

## 2. Processor Hardware Design

The preceding paragraphs described the functional design of the instruction execution cycle, and the control philosophy for the DP M PE processor module. This subsection presents the hardware block-level design and complexity estimates for the processor. The processor is composed of four basic units as shown in Figure 45. These units are the instruction register select and decode, processor control, micro data processor, and status register.

### *a. Instruction Register, Select, and Decode Block*

The instruction register, select, and decode block (Figure 54) is composed of the instruction register, the instruction decoder, and the register select multiplexer. The instruction register is composed of a single-bit latch for external interrupts, a 10-bit latch for the X, C, and M fields, and two 3-bit up-down counters for the T- and R-fields. The external interrupt is synchronized by latching it into a register when the new instruction is loaded. The counters are used for the multiple word accesses to the register file for double shifts, multiply, divide, and push pop multiple instructions.

The instruction decoder derives its inputs from the instruction register (including the external interrupt flag) and one bit of micro control which is used to force a second operation decode during the execution of standard format instructions. The overflow mask, overflow

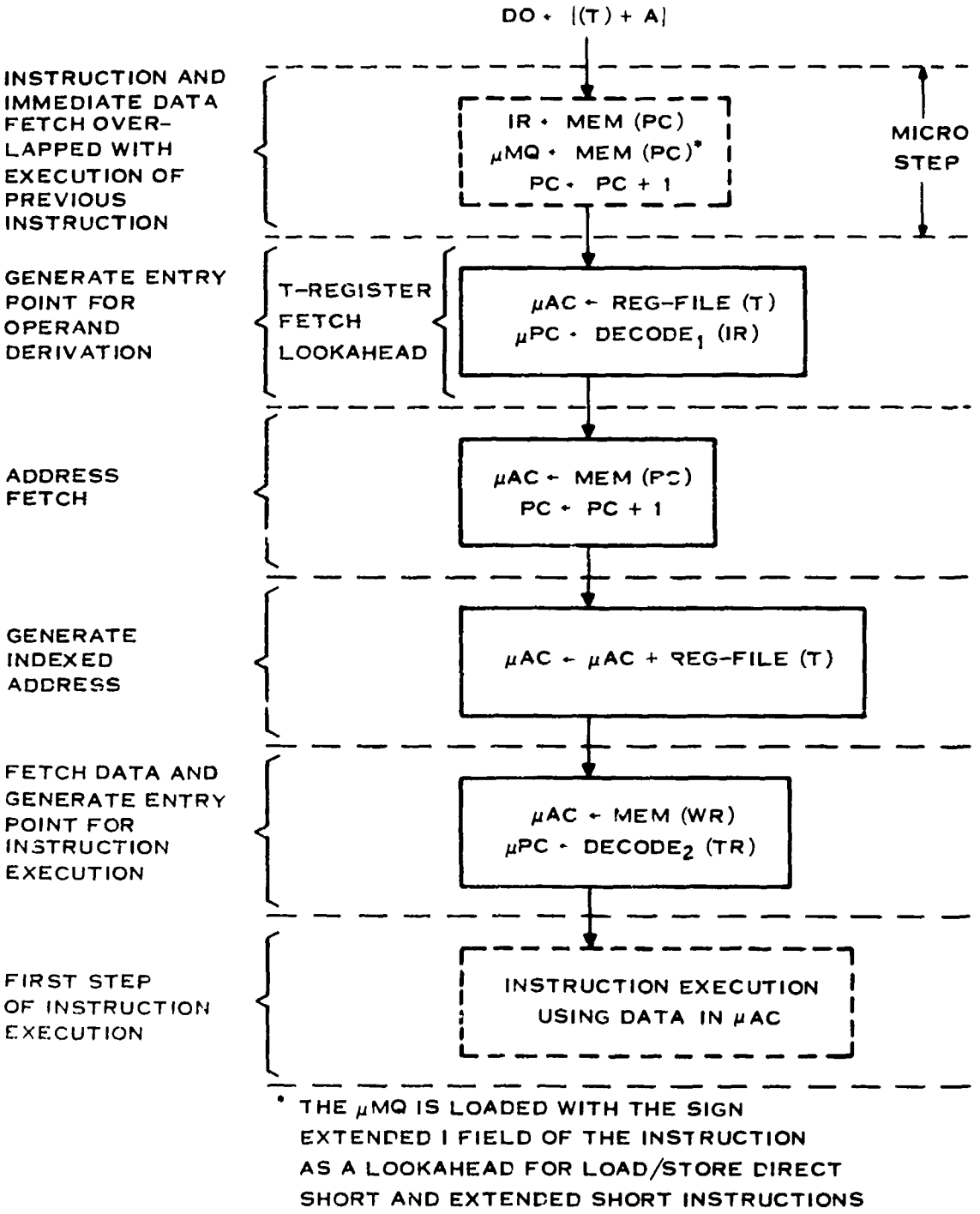



Figure 48 Direct Indexed Operand Derivation

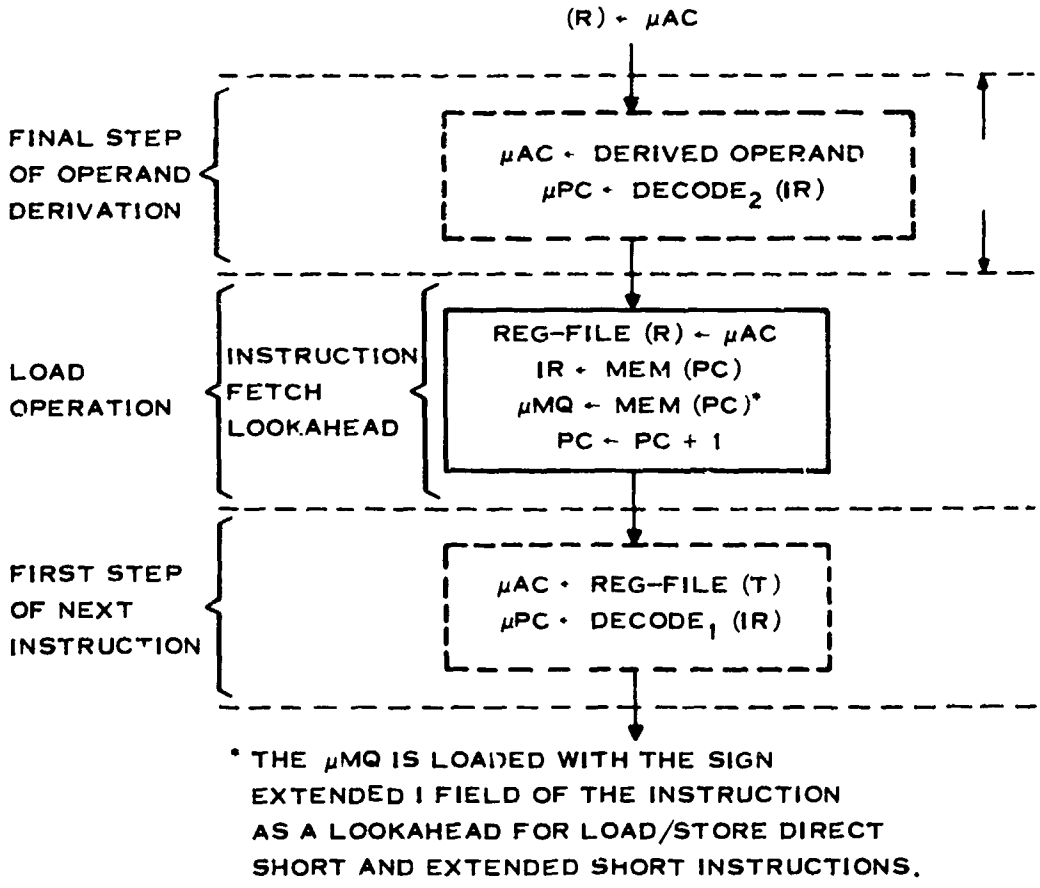



**Figure 49. Load Instruction Execution**

condition, device and memory error masks, I O select, transfer time out, and memory time out flag lines are also input into the instruction decode logic. The instruction decode logic is implemented as a programmable logic array (PLA), has 25 input variables, and generates an 8-bit micro code entry point and a data line into the memory timeout flag. There are six classes of micro memory entry points generated (interrupt, load store direct short, extended short, derived operand, derived address, and standard format operation) that take approximately 80 logical minterms. Lastly, there is the register selection multiplexer which is a dual three-input multiplexer.

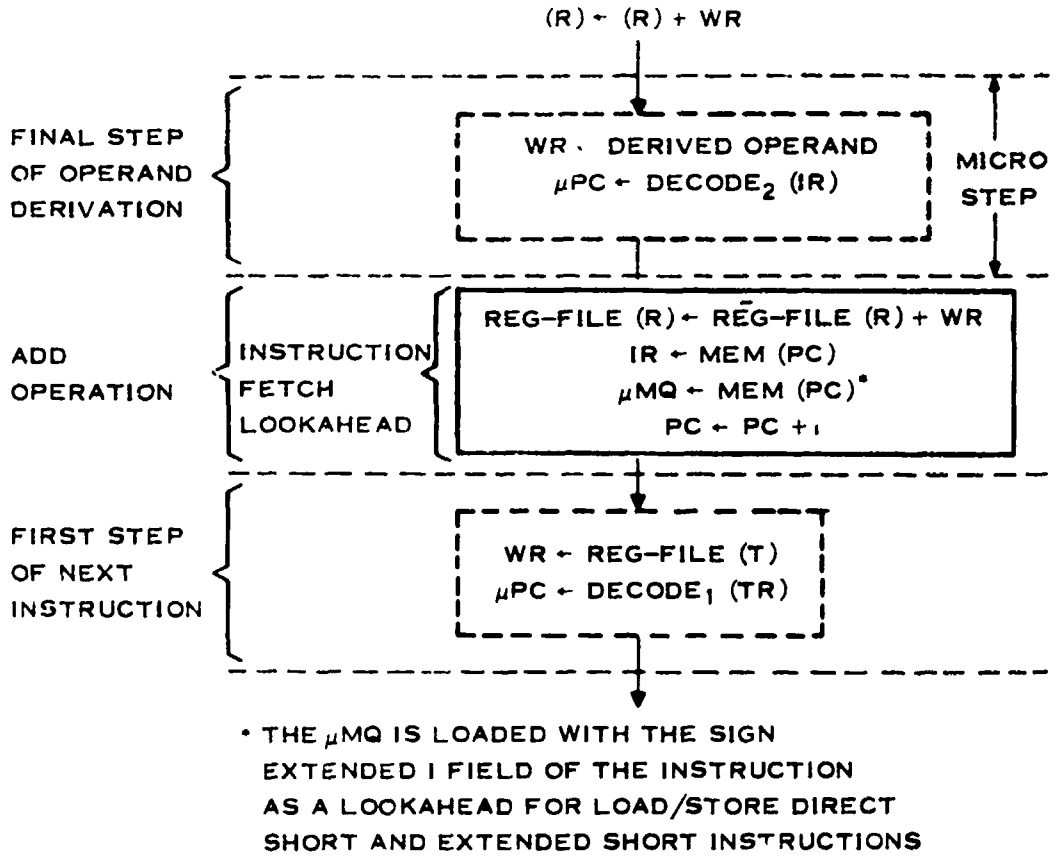



Figure 50 Add Instruction Execution

*b Processor Control*

The processor control is shown in Figure 55. It is composed of the micro sequence control (shown in Figure 56) and the micro memory.

The micro sequencer is an 8-bit-wide microcode address controller. It is composed of an adder, a micro link, a micro address multiplexer, a micro address register and a micro control decoder. The adder is used to increment the micro address register and generate relative branches. The micro link is used to hold a return address for micro looping and micro subroutines. The micro address multiplexer selects the source of the next micro program address. This source can be the instruction decoder, an absolute or relative branch address supplied by the micro code, the next micro instruction in sequence, or the micro link address. Note that the

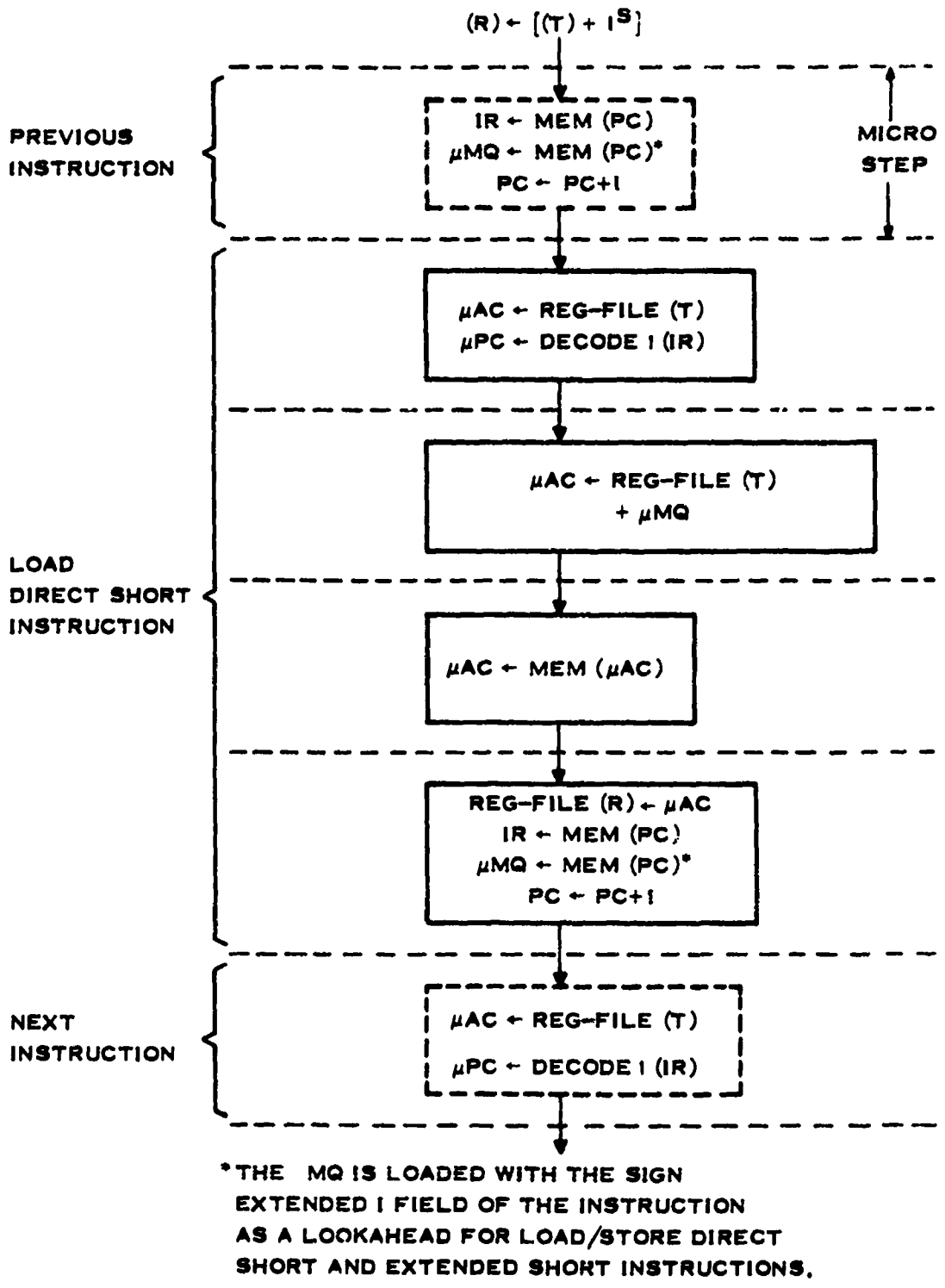



Figure 51. Load Direct Short

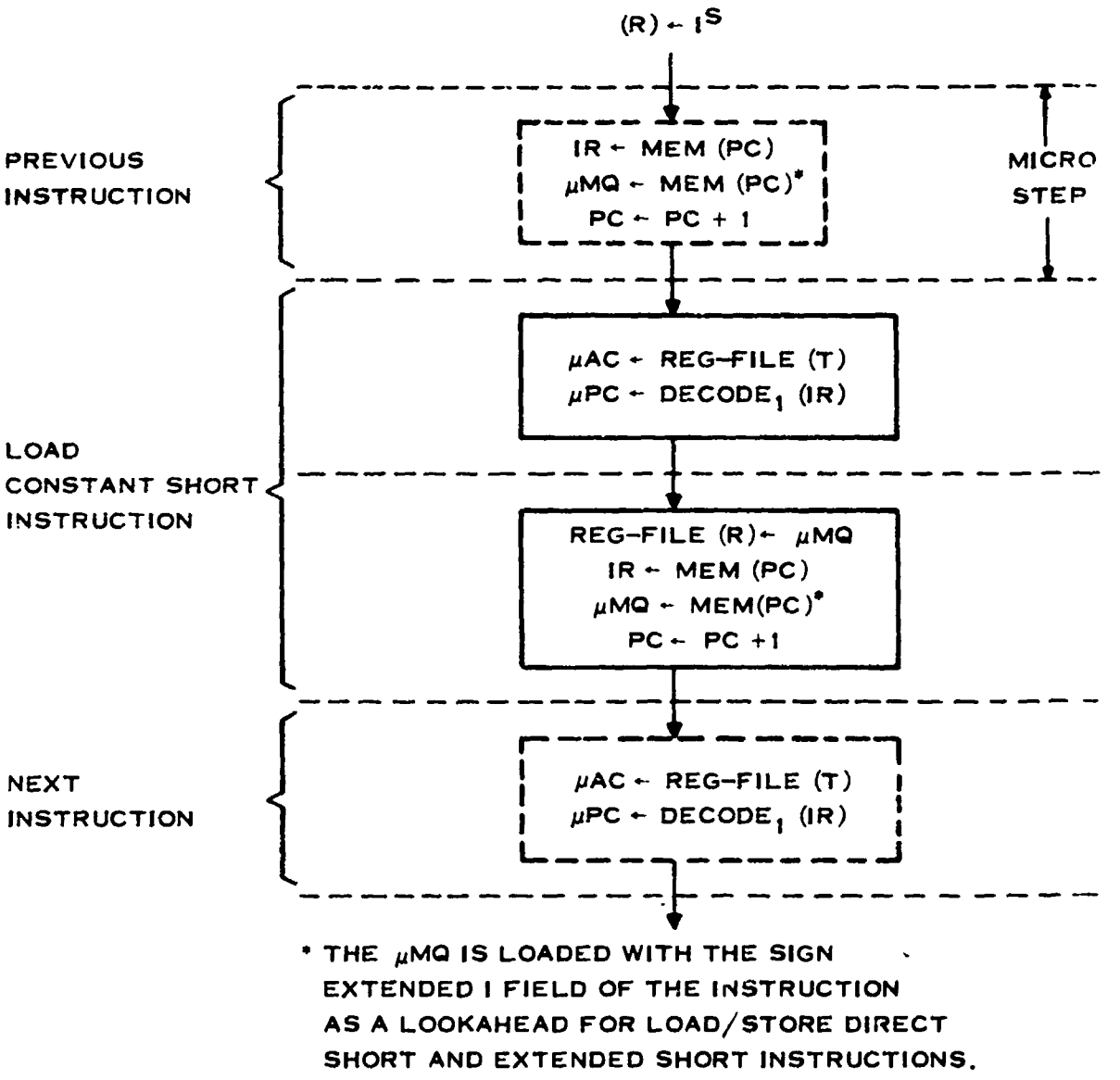
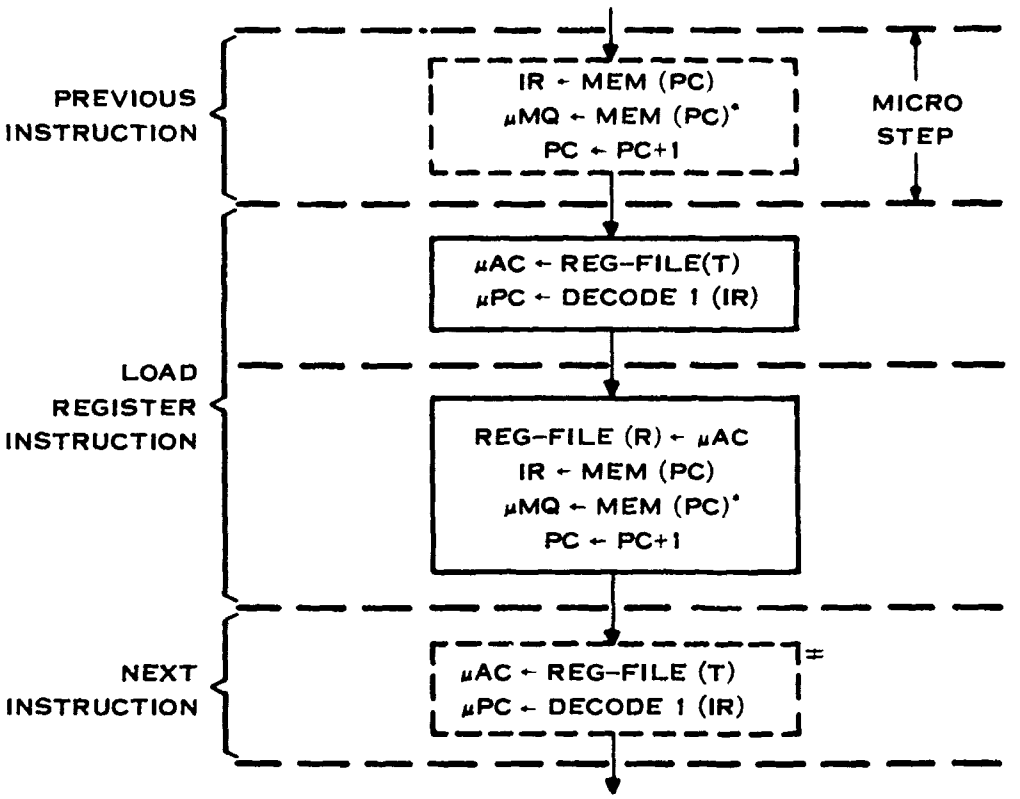



Figure 52. Load Constant Short




*THE MQ IS LOADED WITH THE SIGN EXTENDED 1 FIELD OF THE INSTRUCTION AS A LOOKAHEAD FOR LOAD STORE DIRECT SHORT AND EXTENDED SHORT INSTRUCTIONS.

=NOTE THAT THIS EXECUTION IS THE SAME AS FOR ANY OTHER DERIVED OPERAND LOAD INSTRUCTION BECAUSE IT ALWAYS HAS THE DERIVED OPERAND IN THE MICRO ACCUMULATOR (μAC).

**Figure 53. Load Register**

branch address (relative or absolute) shares micro code bits with the other parts of the processor. This reduces the number of micro memory bits while requiring an extra clock in the case of a branch instruction. Because of the instruction decoder for microcode entry points and the micro link register for iterative loop closing, only special cases, such as multiply and divide fixup, require the use of the micro branch. The micro sequence control has its own decode which has 7 control lines from micro memory and 14 internal and external condition lines it can test. This decoder is also a PLA which has approximately 35 logical minterms and 6 outputs which control the adder, multiplexer, and the two registers.

Previous design experiences with instruction complements similar to the DP/M PE indicate that a 256-word control memory will be adequate for the processor micro memory. A

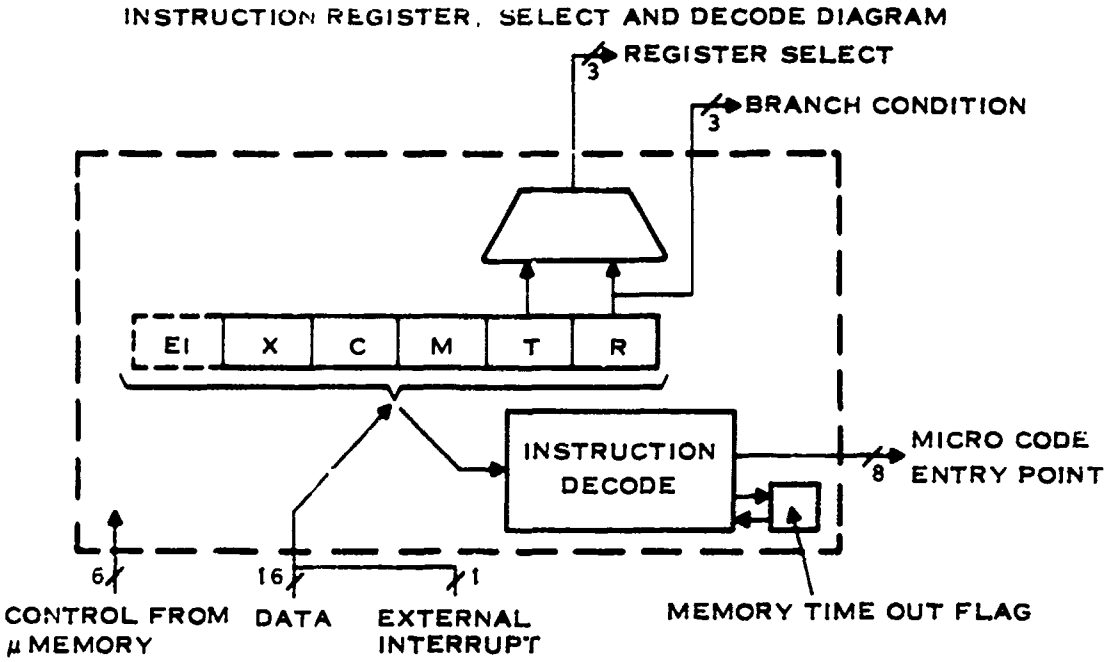



Figure 54. Instruction Register, Select and Decoder Diagram

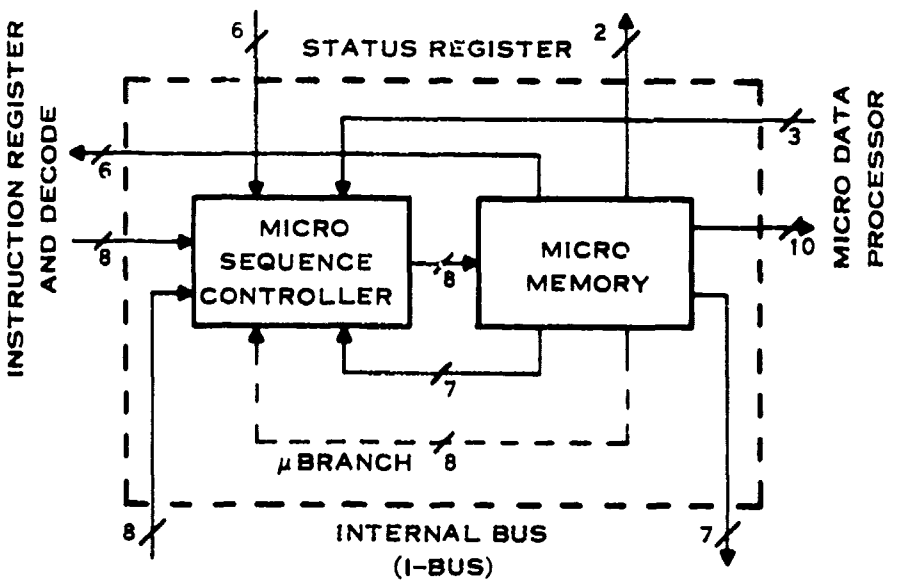



Figure 55. Processor Control Diagram

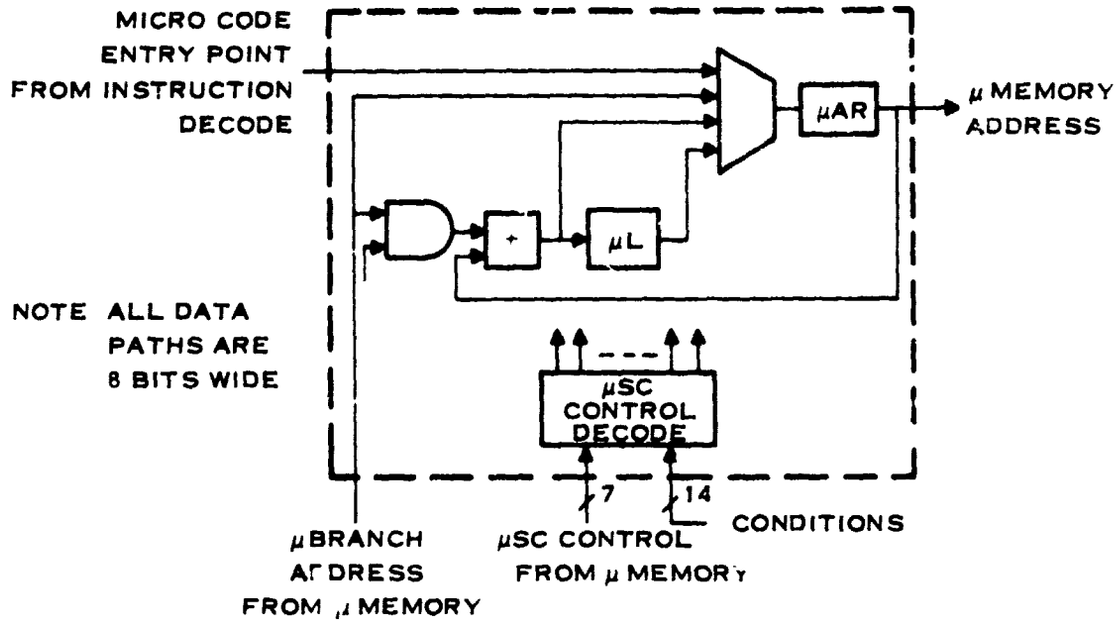



**Figure 56. Micro Sequencer Control (μSC)**

32-bit-wide micro control word will suffice based upon the different local decode sequence. Therefore, the recommended micro memory for the processor is 32 × 256 (8192) bits of ROM

*c. Micro Data Processor*

The DP M PE micro data processor reflects a conventional design approach for a micro controlled processor. It contains its own local memory [the register file (RF)], a program counter (PC) which is a counter rather than a register, an arithmetic logic unit (ALU), and two working registers [the micro accumulator (μAC) and the micro MQ register (μMQ)]. The PE memory and I/O devices are accessed by the processor as if they were externally addressable devices via the "address" and "data" lines. All of the data paths are 16 bits wide except as shown in Figure 57. A few special points worth noting are:

The 6 bits of the internal processor status word are merged together with the 10 external status word bits (brought through the μAC) by the B multiplexer of the ALU. Also, the internal status word is loaded from the output of the ALU when the external status word bits are loaded via the data lines.

The address lines can be driven from either of the two internal working registers, the program counter, or the internal interrupt trap location addresses which are supplied by the micro code. When the address lines are decoded from the μMQ, only the lower eight bits are used since this corresponds to either the directory address of the BILR instruction subroutine or the MIC MOC command address word (CAW).

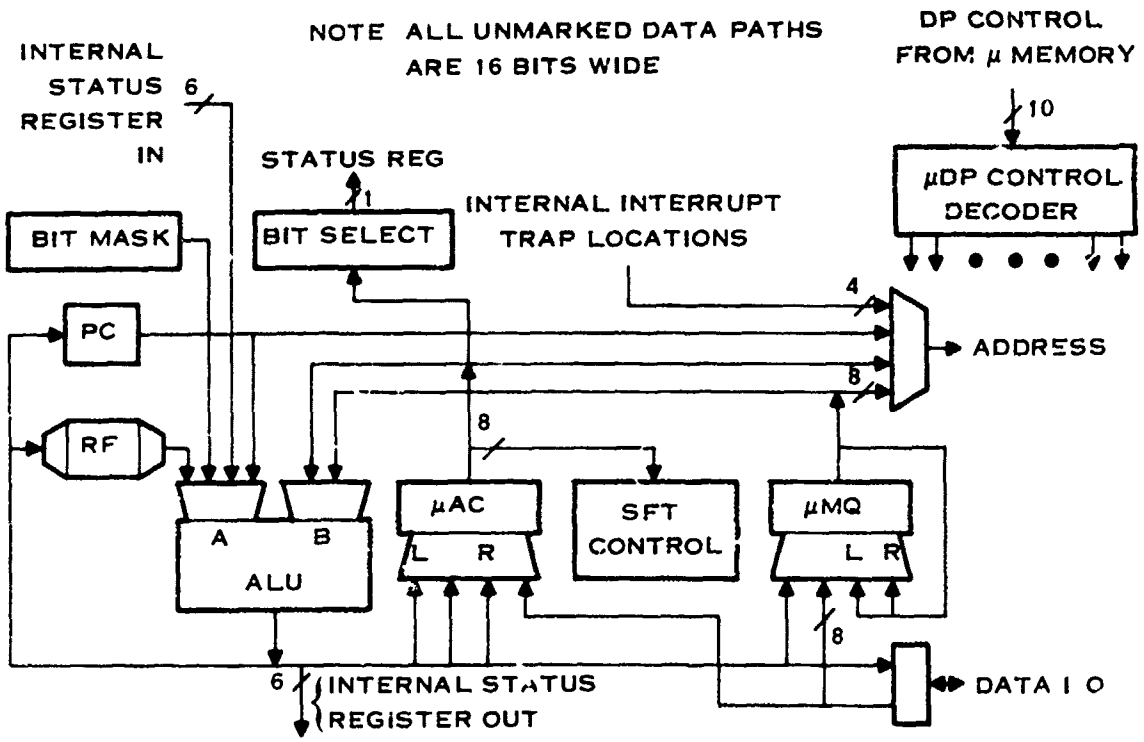



Figure 57. Micro Data Processor

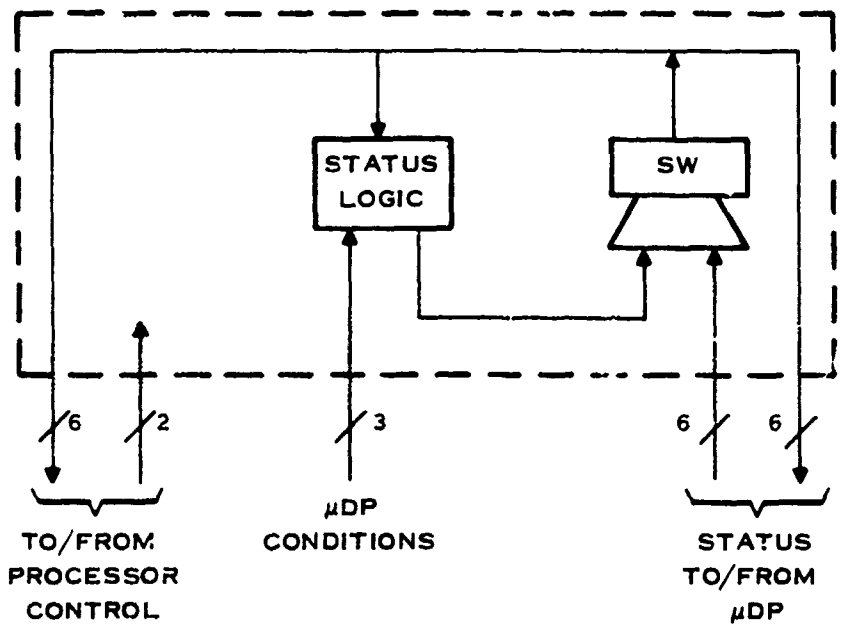



Figure 58. Status Register and Control

The shift count and control are the lower half of the derived operand. This information is loaded from the $\mu$AC which contains the derived operand. This can be done at the same time the $\mu$AC (and possibly also the $\mu$MQ) are being loaded via the ALU with the values to be shifted. Note that the shift iteration counter is also used during multiply and divide instructions.

The $\mu$AC, $\mu$MQ, PC, RF, and data out (as well as the internal status word) can all be loaded with the output from the ALU. In addition, the $\mu$AC can be loaded with the output of the ALU shifted left or right, as well as the data in. The $\mu$MQ can also be shifted (left or right) or loaded with the sign-extended 8-bit immediate data field of the instruction.

The micro data processor control decode has 10 input lines to control the 20 internal gating lines. Based on previous design experience, approximately 100 minterms will be required to implement this control function.

### *d. Status Register and Control*

The status register and its control (Figure 58) is composed of a six-bit register plus the status logic. The status logic has 11 inputs, 6 outputs, and approximately 32 minterms.

## 3. Processor Complexity Estimate

Each hardware block of the DP'M PE processor was examined and an estimate of its approximate device complexity was computed based upon TTL (Transistor-Transistor Logic) equivalent part and device counts. Table 11 summarizes the factors used to estimate processor complexity in terms of:

Logic gates

PLA terms

ROM terms

RAM bits.

**TABLE 11. COMPLEXITY FACTORS**

| GATES(AND, OR, or INVERTER) | GATES |
|---|---|
| Registers and Flip-Flops | 6/bit |
| Counters and Shift Registers | 10/bit |
| Adders | 11/bit |
| Arithmetic Logic Unit (ALU) | 20/bit |
| Multiplexers | lines * (1 + inputs) + 2 $^{log_2}$ (inputs) |
| **PLA** | **PLA TERMS** |
| Programmable Logic Array (PLA) | minterms * (2 * inputs + outputs) |
| **ROM** | **ROM TERMS** |
| Read Only Memory (ROM) | words * (address lines + outputs) |
| **RAM** | **RAM BITS** |
| Register Files and Read/Write Memories (RAM) | 1 bit |

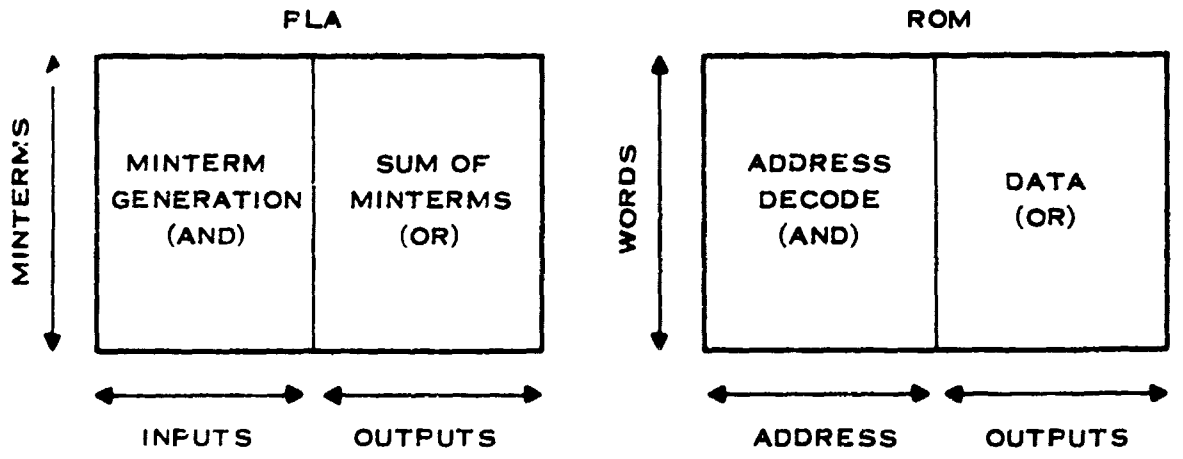



**Figure 59. PLA Versus ROM Layout**

Each of these descriptive terms is relatively technology-independent and, consequently, has been selected to compute the processor complexity. As background information, a brief review of the impact on device complexity of PLAs and ROMs is given.

The semiconductor device construction of a PLA and ROM are quite similar (see Figure 59).

In the case of the ROM, all possible minterm combinations of the address lines are generated, i.e., address decode (minterms = words = $2^{\text{address lines}}$). The data read from the ROM corresponds to selecting those minterms (words) that are ORed together to form the output (bits). The device area for an ROM can be divided into the address decode (word select) area and the data area. A rough approximation of the address decode area can be estimated by taking the number of address lines (true and complement) times the number of words (minterms). This would lead to an area that would correspond to:

$$\text{Area} \sim 2 * \text{address lines} * \text{words}$$

The multiplicative factor of 2 is used to account for both the true and complement of each address line that must be used for address decode. Since all possible addresses (minterms) must be generated and there is no requirement for programming the address decode logic, this can be accomplished efficiently. This configuration permits approximately a two-to-one reduction in the area required for the address decode. Similarly, the data area can be estimated by taking the number of words times the number of outputs; i.e., the data area is proportional to the number

of bits. Since the data area of the ROM must be programmable and can form any pattern, there is not the corresponding area reduction possible as with the address decode. The area of an ROM is thus going to be measured in "ROM TERMS" which is a measure of the word decode area and the data area

$$\text{ROM TERM} = \underbrace{\text{word}}_{\text{bits}} * (\text{outputs} + \text{address lines})$$

Typically, the number of address lines is nearly equal to the number of output lines; therefore, the area for the address decode is about the same as the area for the data.

The PLA is similar to an ROM except not all possible minterms (words) are generated, and the minterms to be generated are programmable. Thus, both the minterms and the sum of minterms are programmable. This structure permits random logic to be mask-programmable, as is the data in a memory. Since the minterms are also programmable, there is not the two-to-one reduction in decode area, as there was in the case of the ROM. The PLA terms must include the two-to-one expansion of the input lines into the true and complement signals. The equivalent area factor for a PLA term is

$$\text{PLA TERM} = \text{minterms} * (\text{outputs} + 2 * \text{inputs})$$

An estimate of the processor complexity is given in Table 12, including the relative complexity of the various sections of the processor. The total processor device complexity is estimated to be

| | |
|---|---|
| GATES | 1,541 |
| PLA ROM TERMS | 22.2K |
| RAM bits | 112. |

With current TTL technology, there is about a 10:1 ratio between the device area for gates and PLA ROM TERMS, with a RAM bit considered smaller than a gate. Using these guidelines for TTL technology, the processor (areawise) would correspond to 3,873 gate equivalents. These ratios are strongly technology dependent, consequently, this number is not appropriate to use for other than TTL. The PLAs and ROMs tend to dominate the active device count even though they pack very well due to their structural regularity. If desirable, several techniques can be used to reduce the size of the PLAs and ROMs. One method of PLA size reduction is to limit the PLA flexibility by not allowing all inputs to control all outputs. The micro data processor PLA could be cut in half by partitioning it into two sections, each with 5 inputs, 10 outputs, and 50 minterm PLAs. Thus, for a slight reduction in flexibility, substantial chip area reductions can be achieved. Similar techniques can be applied to the ROM program, not all 256 words must have 32 bits. For instance, many of the blocks of micro operations have the same bit pattern on most of their data lines. As an example, AND, OR, ADD, and SUBTRACT operations require identical micro control patterns except for a few of the ALU and status control lines. One of the Texas Instruments micro controlled processors that was recently designed took advantage of this feature by splitting the micro control lines between a 32-word ROM and a 256-word ROM. The 32-word ROM controlled the majority of the micro control lines while the 256-word ROM controlled the few lines that were unique to a particular operation and or required multiple clock operations. These techniques can be applied to the ROM to reduce its active device area while imposing some loss in flexibility in applying the ROM to processor control operations.

## TABLE 12. PROCESSOR COMPLEXITY SUMMARY

| | | | GATES | PLA Terms (000's) | ROM Terms (000's) | RAM BITS |
|---|---|---|---|---|---|---|
| Instruction Register and Decode | | | | | | |
| Instruction Register | | | 120 | | | |
| EI, X, C, and M Fields | | 66 | | | | |
| T and R Fields | | 60 | | | | |
| Memory Time Out Flag | | | 6 | | | |
| Decode (PLA) | | | | 4.7 | | |
| Inputs | 25 | | | | | |
| Minterms | 80 | | | | | |
| Outputs | 9 | | | | | |
| Register Select | | | 11 | | | |
| | | | 137 | 4.7 | | |
| Processor Control | | | | | | |
| Micro Sequencer | | | 228 | | | |
| Adder | | 88 | | | | |
| Micro Link | | 48 | | | | |
| Micro Address Multiplexer | | 44 | | | | |
| Micro Address Register | | 48 | | | | |
| Control Decode (PLA) | | | | 1.7 | | |
| Inputs | 21 | | | | | |
| Minterms | 35 | | | | | |
| Outputs | 6 | | | | | |
| Micro Memory (256 X 32) | | | | | 10.2 | |
| | | | 228 | 1.7 | 10.2 | - |
| Micro Data Processor | | | | | | |
| Program Counter | | | 160 | | | |
| Register File (7 X 16) | | | | | | 112 |
| ALU and Multiplexers | | | 438 | | | |
| Micro Accumulator and MQ | | | 320 | | | |
| Address Multiplexer | | | 84 | | | |
| Data I/O | | | 50 | | | |
| Shift Control | | | 68 | | | |
| Register (3) | | 18 | | | | |
| Counter (5) | | 50 | | | | |
| Control Decode (PLA) | | | | 4.0 | | |
| Inputs | 10 | | | | | |
| Minterms | 100 | | | | | |
| Outputs | 20 | | | | | |
| | | | 1,120 | 4.0 | | 112 |
| Status Register | | | | | | |
| Register and Multiplexer (6) | | | 56 | | | |
| Status Logic (PLA) | | | | 1.6 | | |
| Inputs | 22 | | | | | |
| Minterms | 32 | | | | | |
| Outputs | 6 | | | | | |
| | | | 56 | 1.6 | | - |
| Processor Total | | | 1,541 | 12.0 | 10.2 | 112 |

Another technique can be applied to the processor to reduce the chip complexity by dividing the processor into two nearly equal devices. This can be done by putting the micro data processor plus the register specification field of the instruction register on one chip and the remaining functions on a separate chip. Using TTL complexity as a basis for estimation, this partitioning yields

| | |
|---|---|
| Data Processor Chip | |
| Micro Data Processor | 1,643 |
| Copy of T&R Fields | 60 |
| Register Select | 11 |
| | 1,703 gates |
| Control Chip | |
| Total Processor | 3,873 |
| Less Micro Data Processor | 1,632 |
| Less Register Select | 11 |
| | 2,230 gates |

This partitioning possibility is shown later in Section VI.

## C. PE MEMORY SPECIFICATION

While there is a fairly clear distinction between the functional and physical architecture of the PE processor, the PE has a relatively simple one-level main memory concept. The design of the DP/M PE memory subsystem considered the following items

Use and type of memories

Size and performance

Protection and error detection

Initialization.

The memory subsystem can be characterized in terms of its likely use by the processor instruction and operational software. The functional use of PE memory can be divided into the following categories

Scratch pad memory (temporary values, files, stacks)

Data (variables and constants)

Program (instruction storage)

The types of main PE memory available are

High-Speed Read Write Random Access Memory (HS-RAM)

Low-Speed Read Write Random Access Memory (LS-RAM) Main Memory

Read-Only Random Access Memory (ROM).

The relationship between the use and the type of memory is shown in Table 13. Two items discussed previously in the processor design are memory type functions the micro program

control store and the hardware register file. In the DP M PE, the high-speed RAM corresponds to the register file which is part of the processor. the lower-speed RAM and the ROM correspond to main memory. The bootstrap loader is a part of the I O subsystem. the processor micro control storage memory and physically write-protected main memory are internal PE units requiring ROM functions

**TABLE 13. MEMORY USE VERSUS TYPE TABLE**

| | HS-RAM | LS-RAM | ROM |
|---|---|---|---|
| Scratch | ✗ | ✗ | |
| Data | | | |
| Variables | ✗ | ✗ | |
| Constants | ✗ | ✗ | ✗ |
| Program | ✗ | ✗ | ✗ |

The recommended size of the DP M memory module is presently 8,192 or 8K words. Past studies indicate a large portion of avionic processing algorithms are amenable to this segment size of computer memory. The limited avionic software sizing analysis that occurred during the design also found little reason to diff r from the 8K memory module specification. As technologies improve and memory devices increase packing densities and reduce cycle times. the 8K module offers the most use in applying DP M to avionic applications. The projected access time for the DP/M PE memory is expected to range from 1 3 to 1 2 microsecond and offers satisfactory PE instruction execution times.

For high-performance processors. it is not uncommon to have the memory segmented into several sections (possibly on a functional basis). In this case. the memory has several access ports. Typically. one port will be assigned to instruction fetching. another to processor data. and the others to high-speed I/O. Projected throughput and partitioning requirements for the DP M system do not justify the use of such a segmented multiple-port memory subsystem. Therefore. the recommended PE memory subsystem has only a single port accessible to users via the internal PE bus.

### 1. Memory Protection

The subject of memory protection mechanisms was reviewed for the DP M design. The requirement for memory protection will depend upon the use of the PE in a given avionic system, and the demand of a given application for prevention of unauthorized writing into read write memory. If a suitable read write memory requires such a protection in the application of the DP M. the following guidelines are suggested.

Memory write protection generally takes one of two forms. The protection can be classified as either word or address protection. In the case of word protection, each word has a bit attached to it which specifies whether it can be written into or not. With address protection, there is some form of address map that specifies which regions of memory are write-protected and which are not. The protected regions are generally specified by either a set of registers which point to the ends of the protected region(s) or a bit map which specifies whether individual blocks (pages) of memory are protected or not. With core memories, it is common to use word protection since there is no performance penalty in reading a word before writing into it. This is not true, though, for semiconductor memories such as will be used for DP/M; it takes nearly twice as long to do a read before write as a simple write This is because there is basically no restore cycle associated with a semiconductor memory. In addition, the structure of the DP/M memory space leads itself to region protection. For the DP/M. low address memory can be used for program and interrupt trap vectors and high address memory can be used for BIU message control. This means that the two ends of the memory space need to be read-only protected

while the center is read/write. Another common situation is where the reverse is true and the center is read-only protected and the ends are read write. Both of these situations can easily be handled with a pair of registers and address comparators. This is the approach that is proposed for the DP/M RAM type memory that requires write protection. This function, if required, will be implemented on the memory control and timing chip.

## 2. Error Detection

Memories normally have some form of error detection and, in some cases, even error correction. The error detection and/or correction is appropriate for the DP/M PE to increase reliability and fault detection. Semiconductor memory errors are normally chip-oriented. That is, a memory error is mainly due to a single chip that has a single bit or a whole data row and/or column that is marginal or has failed. Thus, if a single memory chip contains more than one bit of a word, there is the possibility of multiple bit errors. The recommended approach for DP/M memory is the use of a single parity bit per word. It is also possible to check to see if the correct word has been accessed as well as to see if the data is correct. This can be done by storing the "address parity" along with the data. Further, it is possible to combine the address and the data parity into a single address-data parity bit, and this parity technique is recommended for the DP/M memory.

## 3. Allocation of Memory Space

Figure 60 shows the probable memory address space allocation for the DP/M PE. The processor can address 64K words of memory; however, only a small portion (8K) is expected to be required for operational programs. The low address portion of memory is reserved for the subroutine directory and interrupt trap vector locations. Table 14 lists the likely allocation of memory locations for the PE interrupt trap vectors. A recommended use of high address

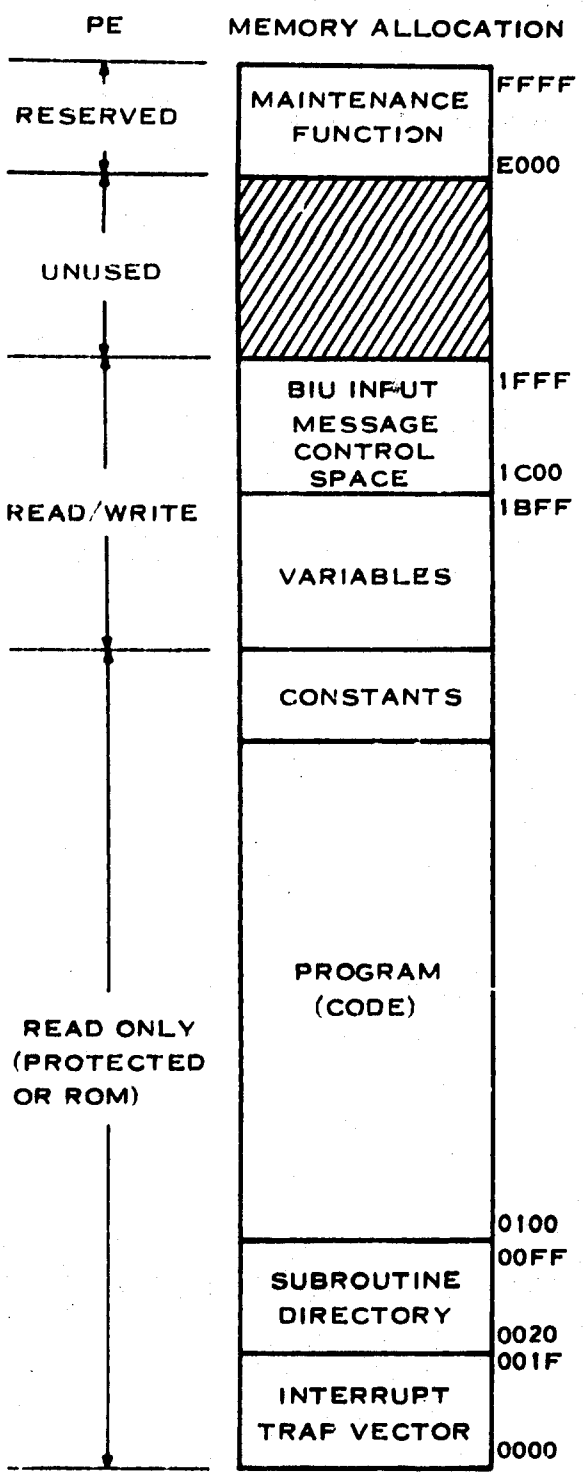



Figure 60. PE Memory Allocation

**TABLE 14 PE INTERRUPT VECTOR SPACE ASSIGNMENTS**

| Location (HEX) | Assignment |
|---|---|
| 0000 0001 | Reserved for diagnostics |
| 0002 0003 | Unassigned |
| 0004 0005 | Invalid instruction |
| 0006 0007 | Overflow |
| 0008 0009 | Device acknowledge error reserved |
| 000A 000B | Memory error reserved |
| 000C 000D | Interrupt 0 |
| 000E 000F | Interrupt 1 |
| 0010 0011 | Interrupt 2 |
| 0012 0013 | Interrupt 3 |
| 0014 0015 | Interrupt 4 |
| 0016 0017 | Interrupt 5 |
| 0018 0019 | Interrupt 6 |
| 001A 001B | Interrupt 7 |
| 001C 001D | Interrupt 8 |
| 001E 001F | Interrupt 9 |

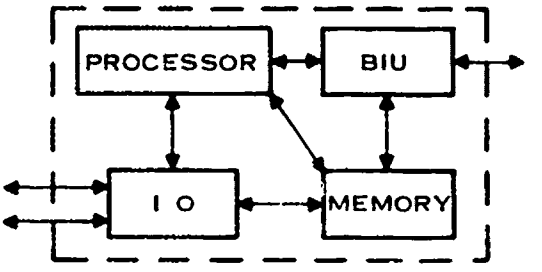



**Figure 61. DP/M PE Sub-element**

memory is for the location of maintenance function software. Subsection IV.I describes how these high order memory locations can be used for such functions as programmer maintenance panel routines or aerospace ground equipment (AGE) maintenance functions.

## D. PE INTERNAL INTERCONNECTION

The four sub-elements of the DP/M PE (shown in Figure 61) require a method of interface for inter-PE communication. The PE internal interconnection must provide facilities for data, address, and various control, status, and clock signals. This requires either several sets of lines, or the time-multiplexing of several of these onto a single set of lines. Multiplexing decreases the number of lines but at the expense of some increase in chip complexity and a corresponding decrease in PE performance. Thus, the use of separate address, data and control lines in the PE is recommended. The alternative is to time-multiplex the address and data over the same lines while keeping the control lines separate. To maintain an adequate performance level and to simplify the intra-PE control, the DP/M PE will use separate address, data, and control lines.

To minimize the number of pins required on the PE sub-elements, a simple internal address and data bus (I-BUS) structure has been used. One of these control signals is used to specify whether a memory or an I/O transfer is in progress, thus permitting the same set of lines to serve for both I/O and memory transfers. Additional control lines allow several different users to share the I-BUS and the use of intra-PE functional modules having different speeds to operate with I-BUS users. Control lines are provided in association with the interrupt, maintenance, and memory control functions. Details of the I-BUS are given in the I-BUS specification and the I-BUS signals are summarized in Table 15. Although there are 80 pins (20 are power and ground) defined for the interface to the bus, only a limited number are used by an particular device. For example, the processor requires an interface with only 48 of the 80 pins.

**TABLE 15. PROCESSING ELEMENT INTERNAL BUS (I-BUS) SIGNALS**

| | Function | Mnemonic | I-BUS* | Processor Module |
|---|---|---|---|---|
| Data and Transfer Control | Data | DATA | 16 | 16 |
| | Address | ADDR | 16 | 16 |
| | I/O Select | IOSL | 1 | 1 |
| | Data Receive | DRCU | 1 | 1 |
| | Transfer Request | TRQ | 1 | 1 |
| | Transfer Acknowledge | TACK | 1 | 1 |
| | Transfer Time Out | TTO | 1 | 1 |
| | Transfer Abort | TABT | 1 | 1 |
| Bus Master Control | Bus Request | BRQ | 1 | 1 |
| | Bus Release | BREL | 1 | 1 |
| | Bus Grant In | BGRI | 1 | 1 |
| | Bus Grant Out | BGRO | 1 | |
| | Bus Master ID | BMID | 4 | |
| Interrupt Control | Interrupt Request | IRQ | 1 | 1 |
| | Interrupt Inhibit | INHB | 1 | - |
| | Interrupt Acknowledge In | IAKI | 1 | |
| | Interrupt Acknowledge Out | IAKO | 1 | 1 |
| General Bus Facilities | Free-Running Clock | FCLK | 1 | - |
| | Master Clock | MCLK | 1 | 1 |
| | Master Stop | MSTP | 1 | - |
| | Clear/Reset | CLR | 1 | 1 |
| | Power/Ground | | 20‡ | 2 |
| Special Functions | Processor Status | PSTAT | 1 | 1 |
| | Memory Parity | MPAR | 1 | |
| | Memory Write Inhibit | MWI | 1 | |
| | Special Functions Line | SFL | 3 | |
| | | | 80 | 48 |

*Normally I-BUS signals are distributed as "low true."

‡For proper power distribution and signal ground returns, there are multiple power and ground pins. In addition, all the standard supply voltages shall be provided to allow mixing of different technologies

A major advantage of this I-BUS structure is the flexibility that it provides. The I-BUS is simply a set of signal lines with a standardized interface specification. This standard interface allows the use of a wide range of different functional modules as well as permitting the upgrading of the system as newer technologies evolve. Figure 62 shows how the processor sub-elements may now be built using current low-power Schottky TTL devices on three small printed-circuit cards. One card would contain the control and the two others would be byte (8-bit) slices of the data processor portion of the processor. As more complex MSI and LSI chips become available, the data portion could be reduced to a single card. Similar situations exist for the memory, I/O, and BIU. The I-BUS structure permits the DP/M PE to be built with present technology, and provides for integrating new implementations of PE sub-elements with minimum impact on existing modules.

A short review of the general capabilities of the I-BUS will serve to illustrate its flexibility. The I-BUS characteristics can be divided into the data and transfer control, bus master control, interrupt control, general bus facilities, and special functions.

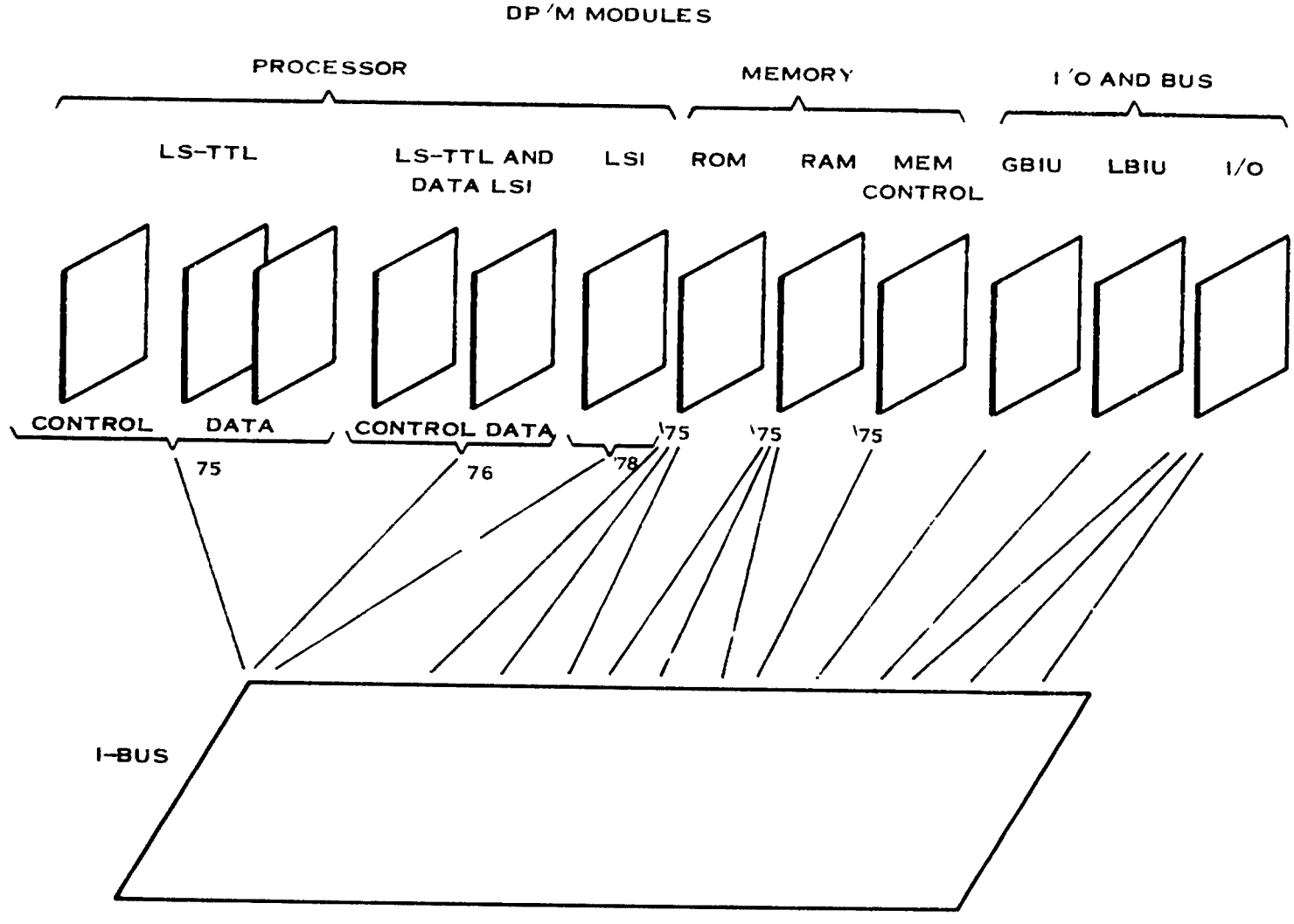



Figure 62. DP/M Modules

## 1. Data Transfer and Control

Data transfers on the I-BUS are effected as a demand/response sequence between a bus master (such as the processor) and a bus slave (such as a memory or an I/O device). Figure 63 shows data being sent from a master to a slave. This activity is initiated by the master via the transfer request (TRQ) line. The slave identifies that it has been specified by decoding the address and I/O-select lines. When the slave has completed its decode process and recorded the data, it responds by asserting transfer acknowledge (TACK). When the master receives the acknowledge, it completes its part of the cycle by terminating its transfer request and releasing the data, address, and I/O select lines. The slave completes its part of the transfer by terminating its transfer acknowledge when it sees the transfer request end.

Figure 64 shows data being received by a master from a slave. This sequence is similar to the send sequence except the slave places the information onto the data lines and then asserts transfer acknowledge. When the master has received the data and latched it into a register, it terminates the transfer by dropping the transfer request line and releasing the address and I/O select lines. The slave then releases the data lines and drops its transfer acknowledge signals.

In addition to this normal transfer request/acknowledge sequence, there are two ways in which a transfer can be terminated abnormally. The first is with the transfer time out line (TTO) which is generated by a watchdog timer on the memory control chip. This time out is used to prevent a master device from locking up the I-BUS by attempting to address either nonexistent memory or I/O devices. When the time out occurs, the master uses the time out signal in the same manner as a transfer acknowledge signal, except it sets its own time out error flag, and proceeds. In the case of the processor, this will correspond to either an I/O acknowledge time out or a memory error. The second abnormal transfer termination is the transfer abort (TABT). This line is used in conjunction with the bus release (BREL) by a higher priority master to force a lower priority bus master to relinquish control of the bus. When this occurs, the lower priority bus master formerly in control simply continues from the point it was when it regains control of the bus. The only effect this has on the pre-empted bus master is that its transfer appears (without time out) to take longer than normal. These "handshaking" sequences for transfers ensure that the data has been correctly transferred independent of the speed of either the master or the slave device. This allows mixing of technologies for memories, processor, I/O, and BIU.

## 2. I-BUS Master Control

There are three control lines and a set of four identification lines associated with bus master control (Figure 65). These three control lines are used to control the assignment of the bus to a bus master. The assignment is performed on a first-come-first-serve basis, with hardwired priority used to resolve simultaneous requests. The four identification lines are associated with the maintenance function. The master that is in control of the bus places its identification code on these lines (processor = 0; the ID default value). This identification is especially useful during the "address breakpoint" maintenance function where it is important to be able to identify who issued the breakpoint address. The use of four lines allows the unique identification of up to 15 bus masters in addition to the processor.

The three bus master control signals are bus request (BRQ), bus release (BREL), and bus grant (BGR). The bus request line is used to initiate a bus master assignment, while the bus release line is used to terminate the assignment. The bus grant line is threaded through each

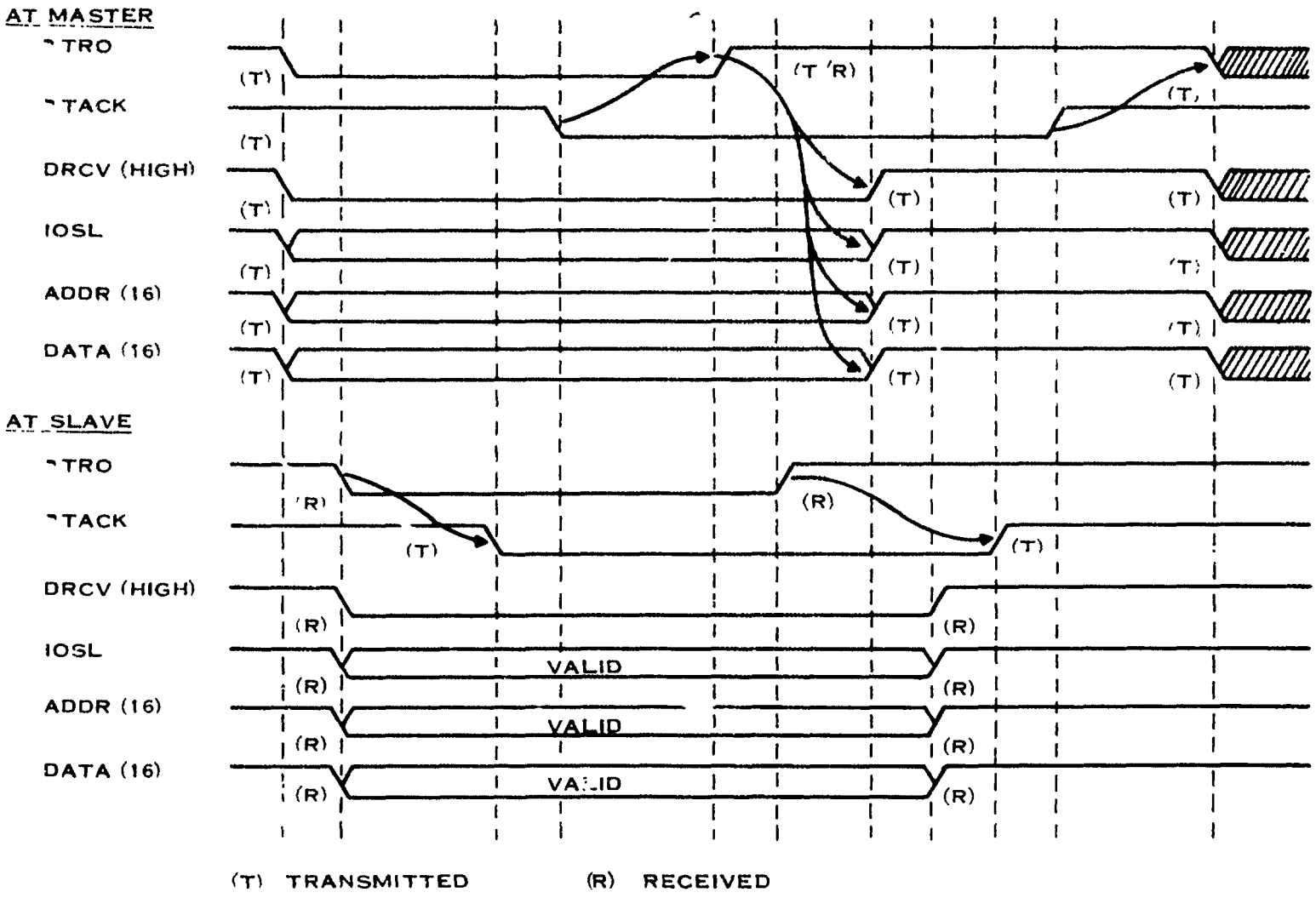



Figure 63. I-Bus Master to Slave (Memory-I/O) Send Cycle (Delay is Exaggerated for Clarity)

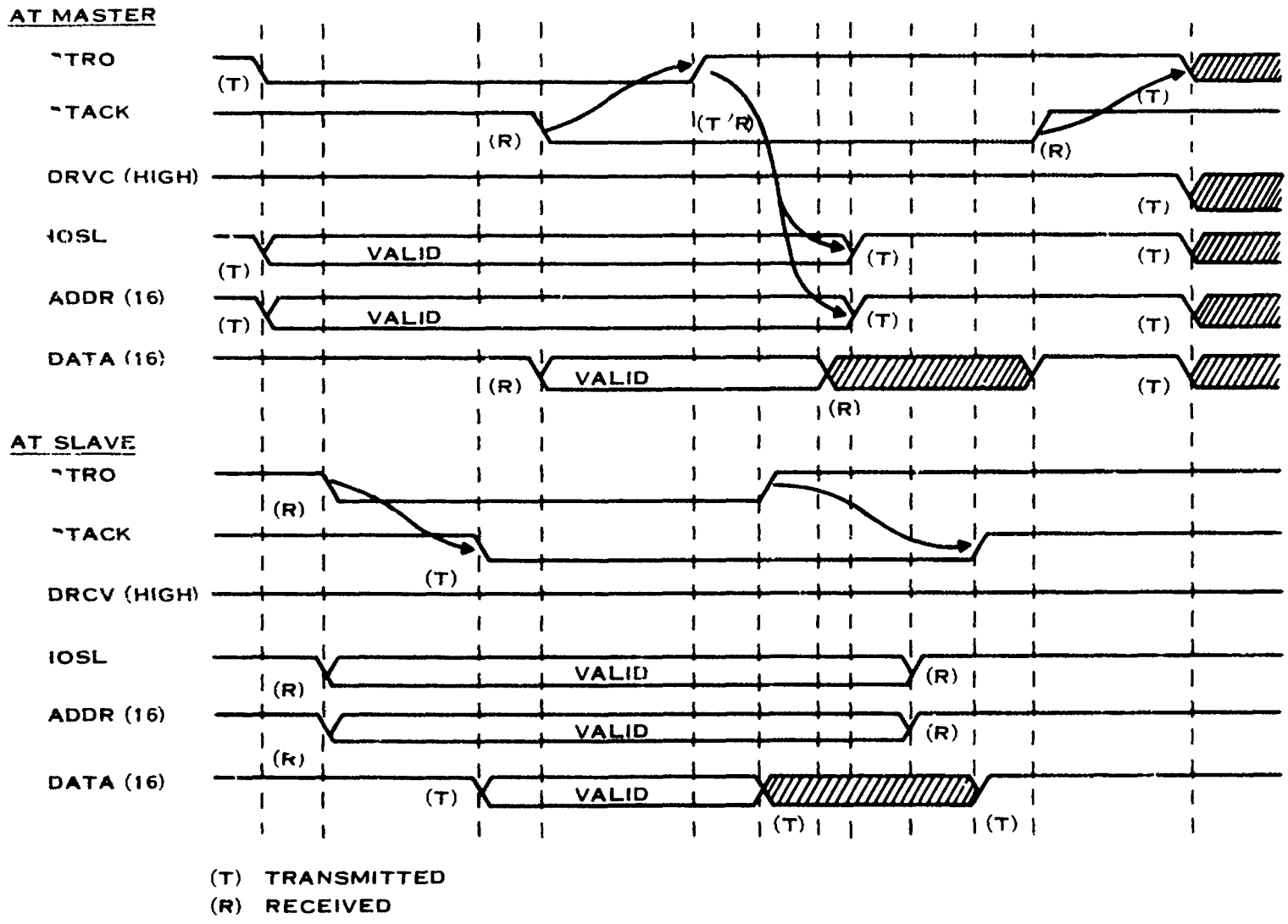



Figure 64. I-Bus Master From Slave (Memory-I/O) Receive Cycle (Delay is Exaggerated for Clarity)

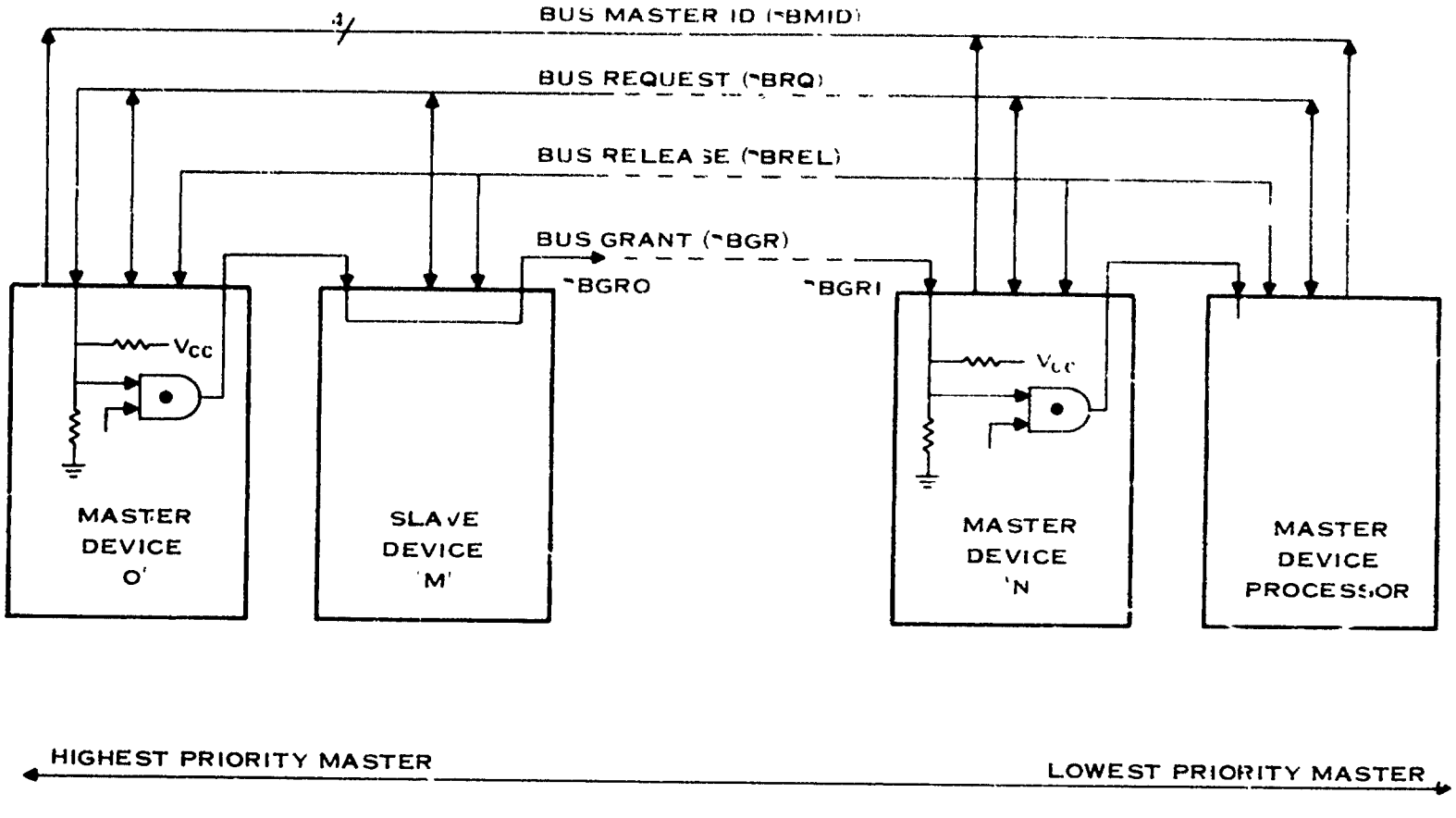



Figure 65. I-Bus Master Device Interconnection

module and is used to resolve conflicts when two or more masters request a bus simultaneously. Figure 66 shows a representative bus-assignment sequence. When the bus quiescent master "A" requests the bus by asserting bus request line, no other master may request the bus. When the bus request line is asserted, the bus grant signal starts propagating through the modules on the I-BUS. When the signal reaches the highest priority requesting module, it activates that bus master, which blocks the bus grant signal from propagating to any lower priority module which may also be requesting the bus. At this point, the bus master (which has been selected) places its identification on the bus master ID lines (BMID) and starts transferring. When it starts its last transfer (depicted by the X-X vertical line) it issues a bus release. This command forces all bus masters including itself to remove their bus requests. As soon as the bus request(s) have been removed, bus release is also removed. When the bus release has been removed, any master can request the bus by requesting the bus request line. Again, the bus grant signal will propagate to the highest-priority requesting bus master. This new bus master will wait until the bus is quiescent (transfer request and transfer acknowledge have ended) and start its transfer cycle(s). This sequence allows bus mastership resolving to be overlapped with the data transfers, thus minimizing or partially eliminating the overhead associated with bus mastership assignment. As was noted under the data transfer discussion, a higher-priority bus master can force a lower-priority master off the bus by pulling on the bus release line which will force all bus masters to release the bus and remove their bus requests.

### 3. I-BUS Interrupt Control

There are three I-BUS control lines associated with interrupt control. They are interrupt request (IRQ), interrupt inhibit (INHB) and interrupt acknowledge (IAK). Interrupts are serviced on a first-come-first-serve basis with priority resolution if there are simultaneous interrupts. The interrupt request is issued by any device that wishes to interrupt the processor (Figure 67). There may or may not be an interrupt mask associated with (and internal to) the device which will inhibit the device from issuing an interrupt request. In addition, there is an interrupt inhibit line which is used by the maintenance panel to inhibit all devices from interrupting the processor when it is operating in the maintenance panel mode. When the processor completes the execution of the current instruction, it honors any interrupt by asserting interrupt acknowledge. The interrupt acknowledge line is threaded through all modules and activates the highest-priority interrupting device. This device inhibits the interrupt acknowledge (IAK) signal from propagating to lower-priority modules. When the device receives its interrupt acknowledge, it places the address of its memory interrupt trap vector location onto the data lines. It then indicates that it has placed this address on the lines by asserting transfer acknowledge. The processor latches the address into its internal register and completes the cycle by removing the interrupt acknowledge. The device will then remove its interrupt request when it asserts transfer request and removes the address from the data lines and terminates its transfer acknowledge when the interrupt acknowledge ends.

### 4. General I-BUS Facilities and Special Functions

The general bus facilities are composed of a free-running clock (FCLK), a master (MCLK), a master stop (MSTP), a clear reset (CLR) signal, power and ground. The free-running clock (FCLK) can be used by any device that needs a clock signal. It does not necessarily have any particular timing relationship with respect to any of the other bus signals. The master clock is similar to the free-running clock except that it is controlled by the master stop signal while the free-running clock is not. In general, the free-running clock will be used by slave devices, while

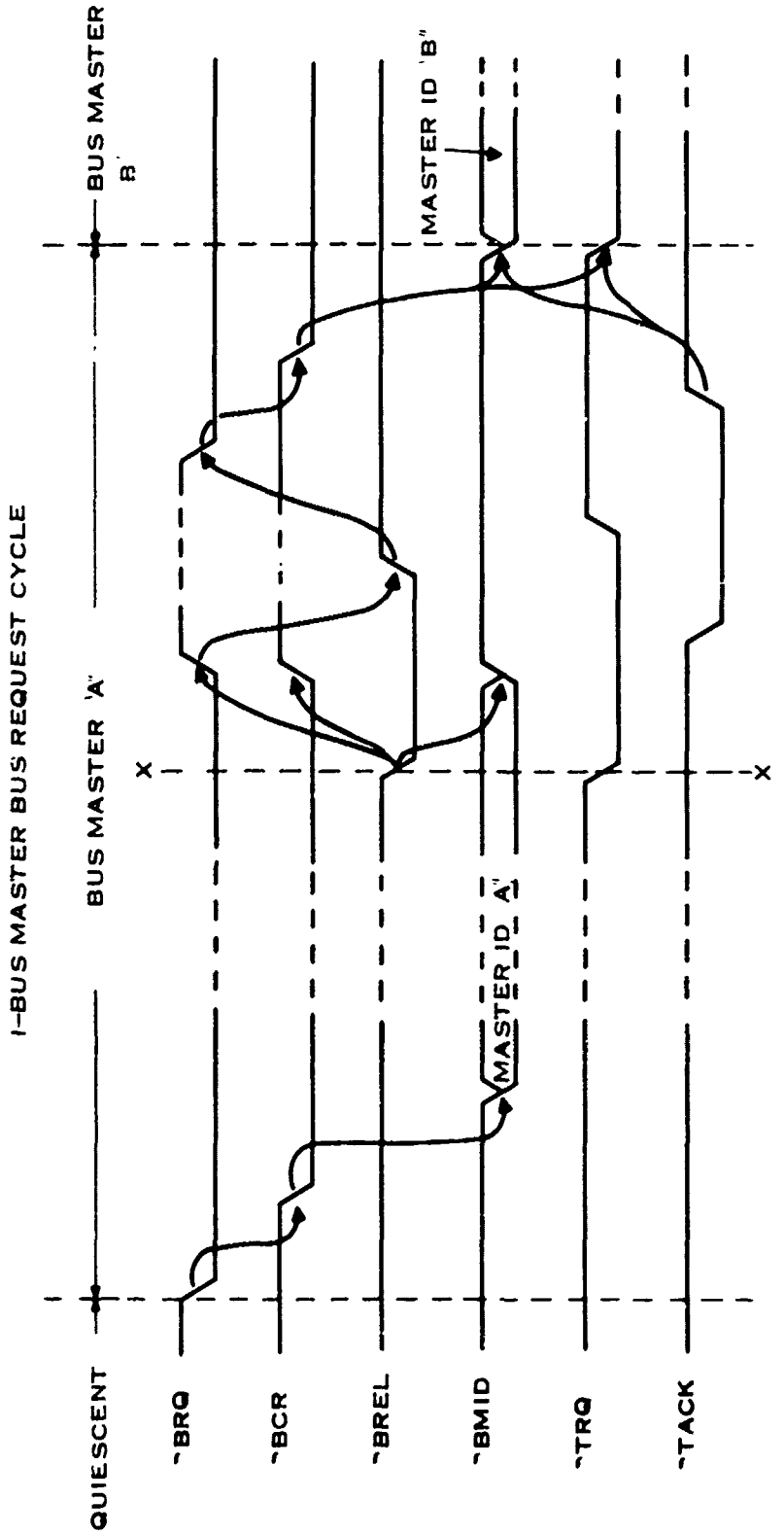



Figure 66. I-Bus Master Bus Request Cycle

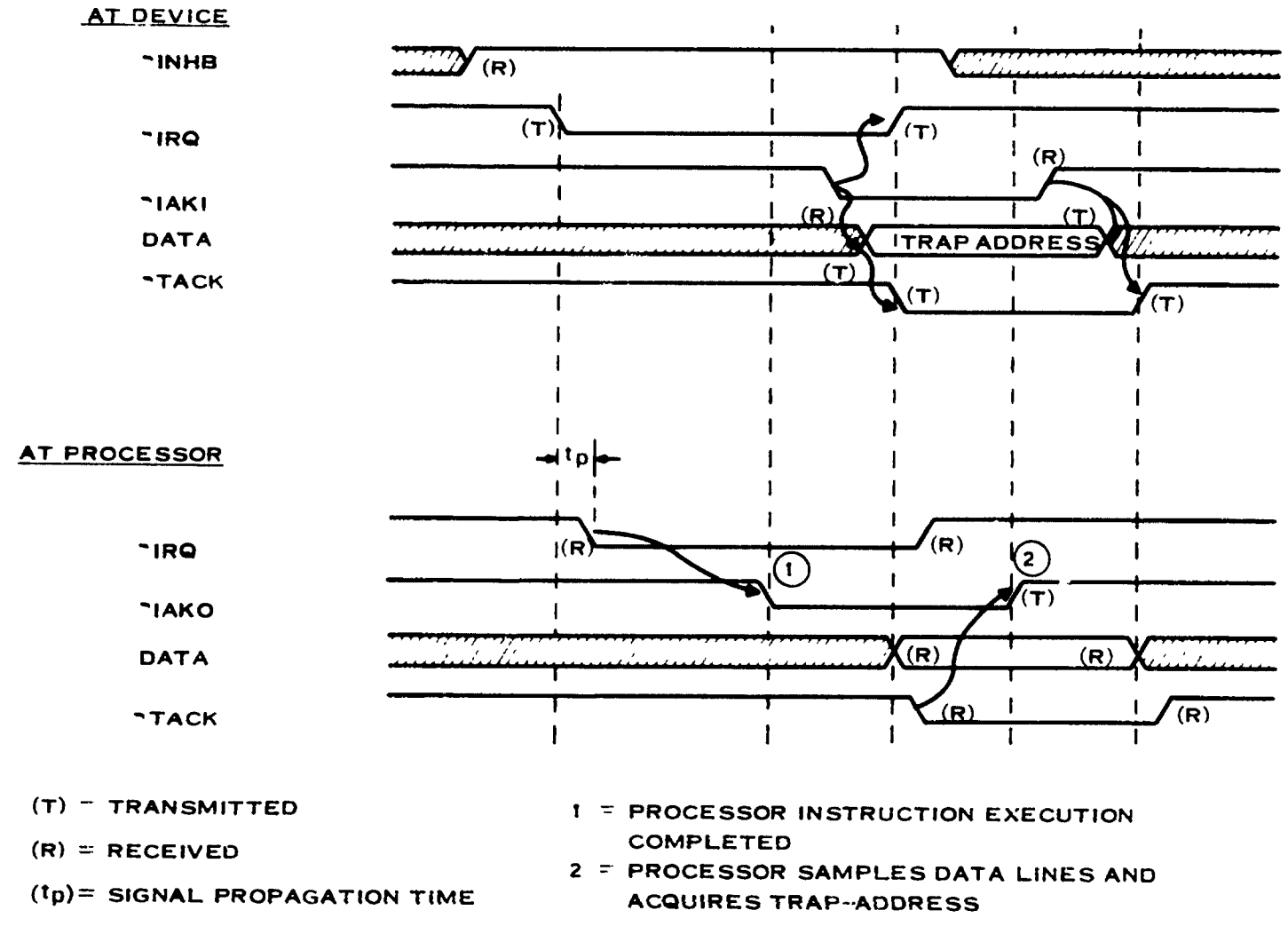



**Figure 67. Interrupt Interface Sequence**

## E. MAINTENANCE DESIGN CONSIDERATIONS

One phase of the DP/M design effort examined the impact of the likely maintenance functions to be required of the PE in its role as an avionics processing unit. This examination included both the hardware- and software-related aspects of maintenance. The following subsections discuss the key points identified for PE hardware check-out, built-in-test (BIT) concepts, and a programmer maintenance panel interface concept.

### 1. PE Hardware Checkout

Since the processor portion of the PE is likely to be a single LSI chip, or possibly a few devices, there is little point in attempting to isolate hardware faults below the major PE sub-element (Processor, Memory, BIU, and I/O). Thus, for hardware checkout, it would be adequate to provide a maintenance capability which is based upon a PE memory and I/O extension and use a single go no-go BIT software program. The following approach is proposed. The processor can be tested by initiating an interrupt that causes the processor to execute its BIT diagnostic program. If the processor is operating correctly, it will execute its program and send a "status good" complete signal back. If the processor has failed, the signal will not be sent, thus indicating that it has failed. Once the processor has shown that it is operating correctly, it can be used in conjunction with additional test programs to test the remaining sub-elements.

### 2. Built-In-Test (BIT) Concept

The functional design and architecture of the PE allows the system user a choice as to the level of BIT to be used. This section highlights one approach to BIT go/no-go testing. The key features of this BIT approach are:

- Control of the internal bus memory address and interrupt line to force the processor to execute a BIT software routine
- A small and thorough software BIT diagnostic routine that performs processor and memory self-test
- Minimum external control circuitry that is required to initiate and monitor the BIT results.

This BIT approach has an added advantage in that the same testing philosophy can be used for flight-line, in-flight, and AGE diagnostic environments.

The key to this testing philosophy is the common internal bus (I-BUS) that interconnects the PE sub-elements and allows data transfers between memory, processor, and input output devices. By proper control of the memory address and interrupt lines, the processing unit can be directed to execute the BIT software routine in the same manner as a normal interrupt routine. Manipulation of these I-BUS control lines can be via external AGE circuitry through an AGE connector on the PE or by similar circuitry within the PE. Previous applications of this BIT concept used a special operator (pilot) control line to initiate the BIT. The choice of implementation and degree of sophistication is an option of the system designer.

When the PE starts execution of its BIT routine, a watch dog timer can be used to verify that execution of the BIT is within allowable time limits. If, in the course of running diagnostic

## E. MAINTENANCE DESIGN CONSIDERATIONS

One phase of the DP/M design effort examined the impact of the likely maintenance functions to be required of the PE in its role as an avionics processing unit. This examination included both the hardware- and software-related aspects of maintenance. The following subsections discuss the key points identified for PE hardware check-out, built-in-test (BIT) concepts, and a programmer maintenance panel interface concept.

### 1. PE Hardware Checkout

Since the processor portion of the PE is likely to be a single LSI chip, or possibly a few devices, there is little point in attempting to isolate hardware faults below the major PE sub-element (Processor, Memory, BIU, and I/O). Thus, for hardware checkout, it would be adequate to provide a maintenance capability which is based upon a PE memory and I/O extension and use a single go no-go BIT software program. The following approach is proposed. The processor can be tested by initiating an interrupt that causes the processor to execute its BIT diagnostic program. If the processor is operating correctly, it will execute its program and send a "status good" complete signal back. If the processor has failed, the signal will not be sent, thus indicating that it has failed. Once the processor has shown that it is operating correctly, it can be used in conjunction with additional test programs to test the remaining sub-elements.

### 2. Built-In-Test (BIT) Concept

The functional design and architecture of the PE allows the system user a choice as to the level of BIT to be used. This section highlights one approach to BIT go/no-go testing. The key features of this BIT approach are:

- Control of the internal bus memory address and interrupt line to force the processor to execute a BIT software routine
- A small and thorough software BIT diagnostic routine that performs processor and memory self-test
- Minimum external control circuitry that is required to initiate and monitor the BIT results.

This BIT approach has an added advantage in that the same testing philosophy can be used for flight-line, in-flight, and AGE diagnostic environments.

The key to this testing philosophy is the common internal bus (I-BUS) that interconnects the PE sub-elements and allows data transfers between memory, processor, and input output devices. By proper control of the memory address and interrupt lines, the processing unit can be directed to execute the BIT software routine in the same manner as a normal interrupt routine. Manipulation of these I-BUS control lines can be via external AGE circuitry through an AGE connector on the PE or by similar circuitry within the PE. Previous applications of this BIT concept used a special operator (pilot) control line to initiate the BIT. The choice of implementation and degree of sophistication is an option of the system designer.

When the PE starts execution of its BIT routine, a watch dog timer can be used to verify that execution of the BIT is within allowable time limits. If, in the course of running diagnostic

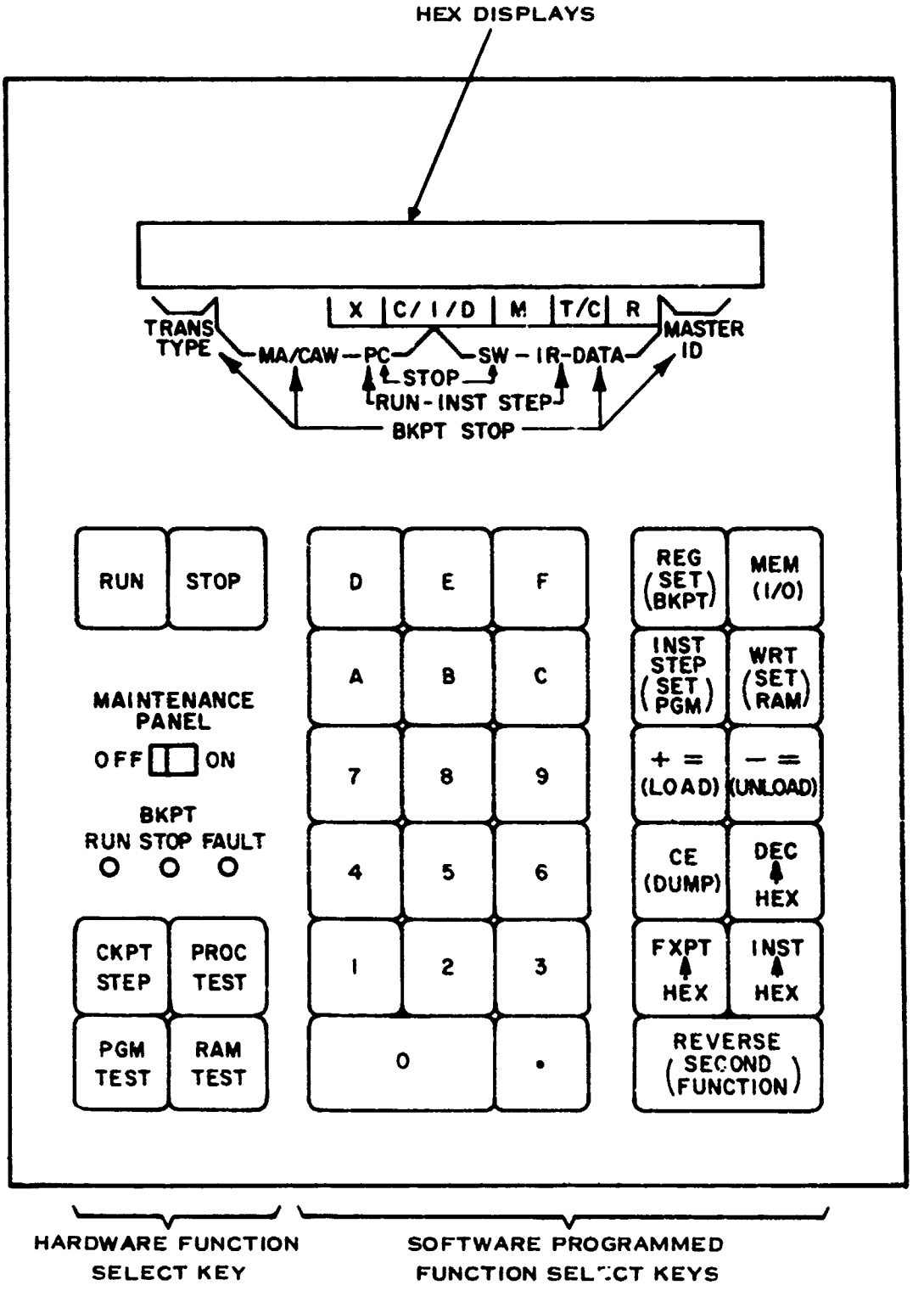



Figure 69. Programmers Maintenance Panel

tests on the processor and memory, an error is detected, an appropriate signal will be generated to the external BIT control circuitry, or the watch dog timer can be allowed to run to completion, signifying a BIT-fail condition.

### 3. Programmer Maintenance Panel Interface

A necessary part of the use of any system similar to DP M is bench-level checkout of the operational software. Normal means of bench-level checkout are provided by programmer maintenance consoles or panels. Often this special test equipment provides limited visibility into the execution of programs and displays results of hardware and software activities via binary or hex data read-out lights. Input data to the maintenance panel likewise is often via toggle or rotary dial switches. One result of the DP M PL design was the identification of a flexible programmer maintenance interface that required minimum external hardware and provided a convenient and adaptable method of loading and monitoring the execution of PL programs.

The proposed programmer maintenance panel consists of a set of hex LED displays, hexadecimal data entry keyboard, mode control pushbuttons, diagnostic invoking pushbuttons, and a series of programmer read modify write function keys. The maintenance panel can also provide a standard teletype interface. Figure 69 shows the maintenance panel organization. Representative software routines provided in the maintenance panel would be:

Program start and stop

Key board control

Display control

Conversion routines Binary to Hex, Hex to Binary, Binary to Decimal, etc.

Memory inspect and change

Register inspect and change

Breakpoint control

Teletype control

Program load and memory register file content display to a hard copy printer device.

The purpose of the maintenance panel is to facilitate system, sub-system, hardware, and software checkout. Its functions are composed of processor and memory hardware diagnostics, program execution display and control features, and a memory address or I/O Command Address Word breakpoint. All of these functions, with the exception of the breakpoint, use the processor itself to perform the necessary operations. This can be accomplished by providing the maintenance panel with suitable interrupt and I/O capability along with its own dedicated section of memory. This section of maintenance memory contains the necessary program and data storage to implement the desired maintenance panel operations. A discussion of likely operation of this maintenance panel concept follows.

Pressing the "STOP" or any of the other test keys on the panel will cause the appropriate interrupt to be generated. Whenever one of the maintenance functions is initiated, the current state of processor execution is saved. This state is restored and execution resumed at the completion of the particular function. Further, while any of these functions or Breakpoint Stop are in progress, all non-maintenance panel interrupts are inhibited via the INHB control line of the internal bus. Functions allowable with this maintenance panel interface include

Hardware diagnostics including processor BIT, memory check-sum, and read/write tests

Program execution display in a user-defined format (hex, octal, decimal)

Program execution control to include instruction step, memory address breakpoint, I/O command address breakpoint, and internal bus step

Program modification to include memory and register content read/modify operations

Programmable interface to a memory load device such as a paper tape reader or magnetic tape cassette or a hard copy printing device.

## F. PROCESSOR/MEMORY DESIGN SUMMARY

This section has described the DP/M processor, memory, PE internal bus, and presented a possible maintenance panel interface concept. It has described their functional and physical characteristics. Various tradeoffs have been made and a final DP/M baseline design presented. This effort included both a top level as well as detailed analysis. The internal bus (I-BUS) structure, one of the most important features of the DP/M architecture, has been defined and described. By defining a standard bus and its interface characteristics the PE architecture is given considerable flexibility and growth capabilities. This allows an orderly growth and evolution of the DP/M PE as new technologies evolve and mature, and as requirements change. One of the applications of this flexibility is the implementation of a simple but powerful maintenance and AGE interface that takes advantage of the processor for execution of test function and requires minimal external memory and I/O circuitry. This concept allows the maintenance/AGE function to be much more flexible and lower cost than normally would be possible with the use of unique special test equipment.

# SECTION V

# EXTERNAL DEVICE COMMUNICATIONS INTERFACE

## A. SYSTEM INPUT/OUTPUT APPROACH

The DP M system approach to providing a communications interface between the DP/M system and the various external aircraft subsystem equipments is directly via the individual DP/M Processing Elements (PEs), as shown in Figure 70. Thus, all external (I/O) device data can be introduced within the DP/M system by being "relayed" through the individual system PEs. This approach, in effect, dedicates devices to PEs but simplifies system bus interface requirements and system hardware overhead. The method of interface chosen is a straightforward parallel digital interface with external devices. PE access to multiple devices or multiple PE access to common devices must be accomplished externally to the DP M system hardware (i.e., all multiplexing functions required must be provided by external hardware interfaces).

The primary advantages of this basic approach are reduced system hardware complexity and simplified system operation. No special bus interface hardware is required for I/O devices and I/O device activities can be placed under the direct flexible control of user programs resident in the PE to which the device(s) is connected.

Intuitive disadvantages associated with this approach were concerned with system reliability or fault tolerance owing to the "dedication" of PEs to I/O devices and the potential increase in Global bus traffic resulting from the distribution of multiple-user I O data. These initial concerns were adequately quashed, with the following realizations:

With respect to overall system reliability, an LSI DP/M PE will possess comparable hardware operational reliability to that of a specialized bus interface for the external device (assuming an I/O distribution bus system approach). The reliability of the device interface, specifically the DP/M PE, is inherently much greater (perhaps by several orders of magnitude) than the majority of avionics devices to which it must interface. Bolstering the reliability of the device would be a fallacious attempt at improving overall system reliability. This alternate approach becomes even more impractical when its implications on added hardware complexity are considered. It is recommended that improved system I O reliability be accomplished in the external equipment (e.g., via redundant devices) where critical reliability requirements prevail.

With respect to potential Global bus traffic increase resulting from the distribution of interfunctional data, it was recognized that most I/O device data tends to be unifunctional in usage, i.e., most I/O data is functionally unique and used exclusively by a single function. The bulk of I/O data which is distributed to from the device interfacing PE can be localized and routed via the Local bus links within the affinity group users of the device data. Global traffic increases are expected to be insignificant, with primary bus traffic loading occurring on the Local buses of the "host" affinity group where it is most easily accommodated in terms of bus bandwidth consumption with minimum effect on overall system performance

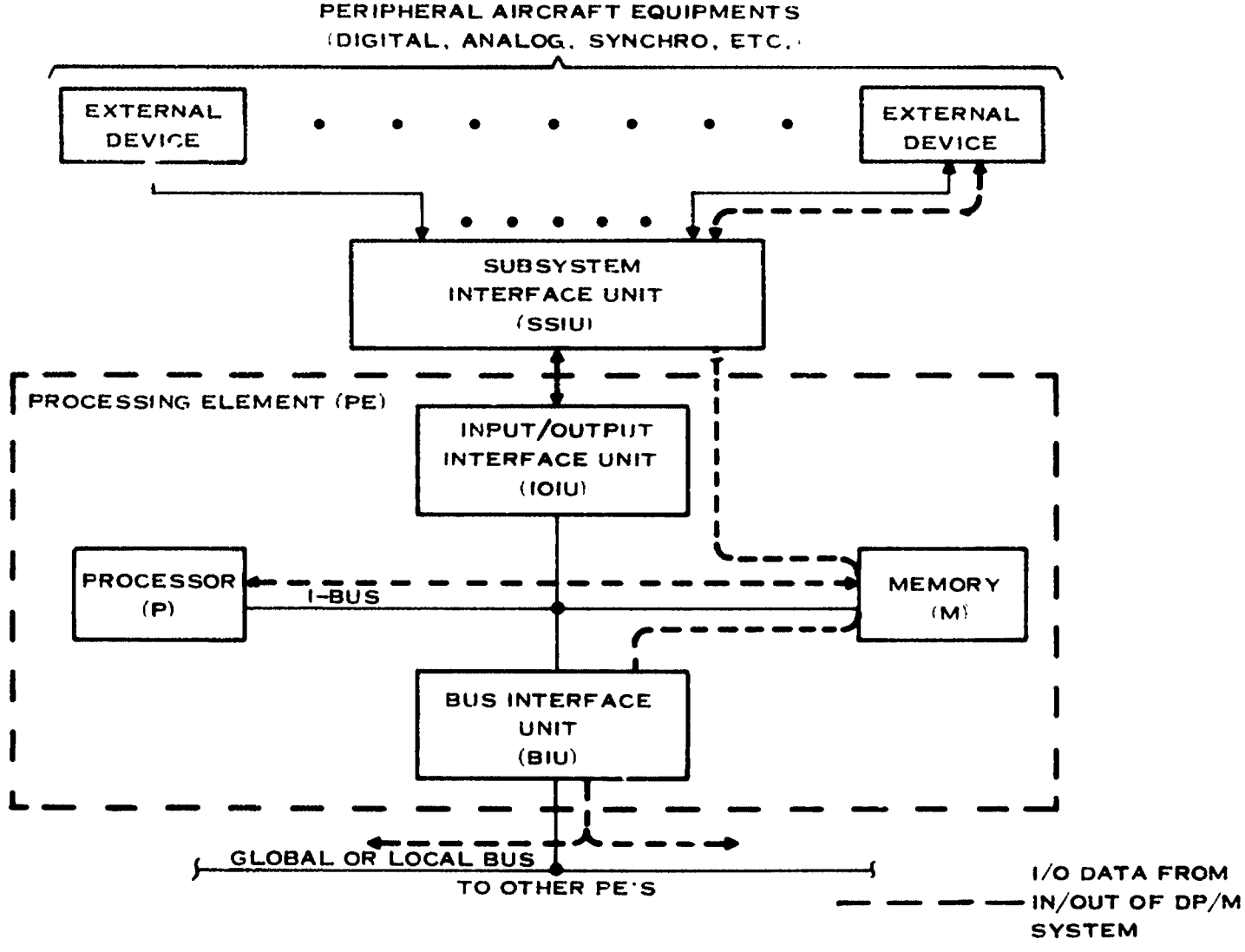



Figure 70. Input/Output Data Interface and System Distribution

In the chosen system approach, the PE can assume the system roles of an external device interface, an external device data preprocessor, or a distributed member of a "centralized" processing function. A PE can assume any or all of the above roles, depending on system requirements. Topological and functional identity boundaries are not necessarily fixed with respect to each other; they may or may not coincide in a particular application.

## B. INPUT/OUTPUT INTERFACE FUNCTIONAL REQUIREMENTS

Having decided upon the aforementioned system-level approach to interfacing the DP/M PE with external devices, an analysis of the actual functional requirements imposed by the envisioned DP/M system I/O environment was performed. In the course of arriving at a "most desirable" PE-to-external device interface design the following basic concepts were considered to be of rudimentary importance in molding the design.

### 1. Standardized Digital Device Interface Compatibility

A "standardized" interface compatibility concept ranks first and foremost as an Input/Output Interface Unit (IOIU) design goal. Since definition of a preliminary standard interface for aircraft equipments and TDM data bus systems is being developed by the Air Force for future avionics applications, the desire to conform DP/M concepts within this standardization framework is intuitively obvious. The subject Air Force standardization approach is presently governed by MIL-STD-1553, "Military Standard for Aircraft Internal Time-Division Multiplex Data Bus." The subject standard external device interface is defined for the Subsystem Interface Unit (SSIU) which interfaces various unique aircraft sensors/effectors with a serial TDM data bus interface unit referred to as a Multiplex Terminal Unit (MTU).

It is believed that an avionic equipment standardization approach sought by the Air Force, as reflected in MIL-STD-1553, is a viable and imminent solution to many present avionic system ailments. Therefore, it has been previously proposed that the DP/M I/O interface be compatible with the SSIU specification. However, such an interface has not been firmly specified/defined and the chance for it assuming a "special-purpose" configuration, tuned to the requirements of the MTU to which it interfaces, are perhaps great. It may be argued that such an interface should be driven by "general-purpose" requirements to allow compatibility with a large number of alternative avionic information processing system elements, one of which is the DP/M PE. It also seems rational and feasible to develop the DP/M input/output interface capability around a similar set of goals to allow maximum compatibility with a variety of equipments. The interface design described herein reflects this goal. Furthermore, it is recommended that the present SSIU interface definition be modified and driven by the same interface philosophy if true aircraft system flexibility is to be achieved in the future (e.g., asynchronous instead of synchronous data transfer protocol).

### 2. Data Rate Requirements

The actual performance potential requirement placed on the I/O interface in terms of data throughput has been initially established by analyzing the I/O traffic experienced in the baseline system mission processing functions. The results of this analysis have shown that total system I/O (external device) traffic is 762,400 kilobits/second. The worst-case maximum for a given function is 522,160 Kbps, this corresponds to the worst-case requirement possible for a single PE, assuming a single PE were assigned to the interfacing of all function-related equipments. This

worst-case data rate of 522.160 Kbps is roughly equivalent to a word-traffic-rate of 32,635 16-bit words/second. Thus, the bandwidth requirements of the avionic system application envisioned do not impose a severe requirement on the throughput capabilities of the I/O interface from an implementation viewpoint.

## 3. Program Interference

Any I/O responsibility associated with a processing function must be considered in the context of program, or processing, interference caused by the I/O data transfers. I/O data rates alone can be misleading unless their effects on processing execution time are considered. The baseline processing functions requiring the relatively higher data rates typically are associated with block transfers of large amounts of data. Thus, to alleviate what may become appreciable I/O overhead execution time in the associated process, it is considered desirable to provide an I/O-processing overlap capability for I/O data transfer operations to reduce processor intervention in these operations to a minimum.

## 4. PE Support Functions

Several existing ancillary functions must be provided within each PE to allow proper PE performance in the DPS environment. In the overall PE design, it is the responsibility of the IOIC to provide these functions. These various functions are individually discussed below.

### *a. Real-Time Reference*

Since the DPS system must operate in a real-time avionics environment, some accurate means of time-keeping must be accessible by each PE. Owing to reasons of system distributiveness, fault-tolerance, and overall performance, it was decided to incorporate a programmable "time-tick" mechanism in each PE. A programmable interval timer was considered most appropriate for this requirement since it allows dynamic time-keeping resolutions, depending on individual application program timing requirements, as well as allowing either "short" time-interval or "time-of-day" real-time-keeping procedures with reasonable accuracy.

### *b. PE Initialization Control*

Control must be provided within each PE to provide an orderly and predictable sequence of PE operations subsequent to the application of regulated DC power to the PE or the occurrence of an externally invoked CLEAR RESET signal. This control places each PE in a known state which accommodates system start-up or initialization activities.

### *c. PE Clock Control*

The IOIC is responsible for generating all clocks required internal to the PE from an externally provided "free" clock source. Clock-gating control provides for proper timing of PE clock generation during power-up conditions and also provides appropriate clock inhibit control in response to external commands (e.g., from a maintenance programmer's control console).

## 5. Implementation Considerations

Consistent with overall DPS PE design goals, the functional design of the I/O interface elements should be amenable to non-complex LSI implementation with low (no) risk

semiconductor technology. Actual electrical characteristics of the interface should abide with the requirements of the standard digital interface (SSIU).

With respect to the actual functional characteristics of the IOIU, it is desirable to design the interface so that a single, common I/O channel, or set of common signal lines, can be used to effect all modes of data transfer operations, i.e., both program (direct) I/O and direct memory access I/O modes. This functional characteristic is of virtue when I/O signal pin-out requirements are considered. In the same context of I/O pin-out frugality, it is desirable to organize the input/output channel as a half-duplex bidirectional ("party line") I/O bus configuration since only half-duplex functional operation is required; however, where special interfaces can dictate nonbus-oriented I/O (i.e., separate input and output lines), the functional design of the IOIU should accommodate this design extension, if practical from an implementation viewpoint.

### 6. Extended Capability Considerations

If the DP/M PE is to become a true universal avionic system building block, its application in nontypical DP/M-like applications should not be precluded. To adapt readily in this environment, the I/O interface should be as flexible (general-purpose) as possible and consideration should perhaps be given to direct PE-to-PE (or processor-to-processor) communications via their I/O interfaces. Therefore, it should be a target of the IOIU design to allow such direct communications via the same I/O channel design as determined by previous considerations.

## C. IOIU FUNCTIONAL DESIGN DESCRIPTION

The IOIU design described herein is functionally represented in the block diagram of Figure 71. This design provides the mechanisms necessary to fulfill the functional requirements outlined above. The operational characteristics of each individual mechanism are discussed below.

### 1. Standardized Data Transfer Channel

The actual "I/O Channel" design chosen for the DP/M PE provides a block-oriented, autonomously controlled direct memory access (DMA) facility for transferring blocks of data between an I/O device and PE memory. Each block transfer activity is initiated under program control but then proceeds autonomously until block completion, at which time an interrupt stimulus is generated to denote data transfer completion. The autonomous channel design was chosen because:

- The SSIU interface inherently requires half-duplex, block-oriented data transfers
- It provides an "overlapped" I/O capability, requiring minimum program execution time overhead
- It does not significantly impact the design with respect to LSI implementation.

This design meets the basic I/O interface conceptual design criteria established from an overall system viewpoint. The operation of each functional element of this design is conventional in nature and is well understood in operational characteristics. The specific signal lines appearing at the physical interface are assigned primarily to provide MIL-STD-1553 SSIU interface functional compatibility and versatility while minimizing pin-out requirements for "single-chip" LSI implementation. In the interest of application and usage flexibility, special operational features have been added to the Data Channel design to allow efficient use with a wide variety

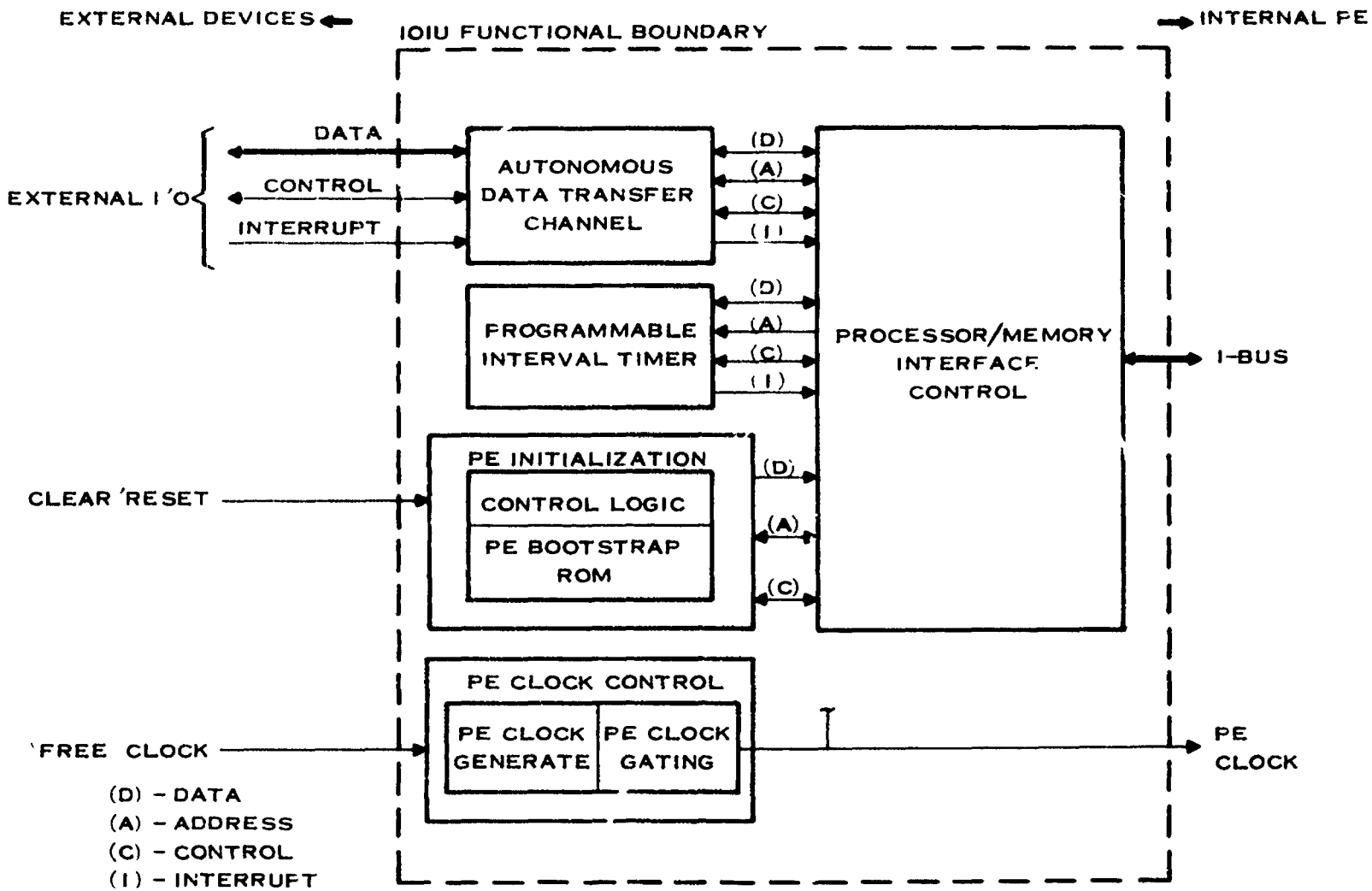



Figure 71. Input/Output Interface Unit Functional Block Diagram

of external equipments. A chaining operation feature has been added to the autonomous DMA operation to lessen program overhead associated with managing block data I/O operations among multiple devices and to facilitate interfacing with multiple data-block oriented devices (e.g., magnetic recording devices which may be used for in-flight recording equipment, mass memory, etc.). The added complexity to the IOIU hardware was judged minimal and should be of no significance in IOIU implementation.

The channel operation allows the discrimination of special "command" information transfers from normal data transfers. Importantly, the transfer of commands and data can be intermixed in a single, common data transfer. This "device-controller-oriented" feature expedites data transfer activity where each transfer of a block of data is preceded by an associated command to the device to be used in interpretation of the data. Such is the case with the standardized SSIU operation, where each block of data transferred is preceded by a control word (to the SSIU) indicating the data transfer activity desired. In addition, it was determined that program-synchronous transfers of single data words (i.e., direct program controlled I/O via instruction execution) is not required in the SSIU interface environment. However, the previously mentioned device controller oriented design feature facilitates "single-word-block" data transfers with only one operation (channel activation). If actual program-synchronous I/O is required in extended applications, these activities can be easily accommodated via direct (buffered) interface with the PE internal bus (I-bus). Thus, the "device-controller" approach was incorporated with a conventional autonomous DMA input/output operational concept in arriving at the final I/O Data Transfer Channel functional design described herein.

The external interface control protocol accommodates asynchronous, fully-interlocked transfers for maximum interface generality instead of the more specially tailored synchronous interface defined in preliminary SSIU specification. An explanation of each interface signal line is presented in Figure 72.

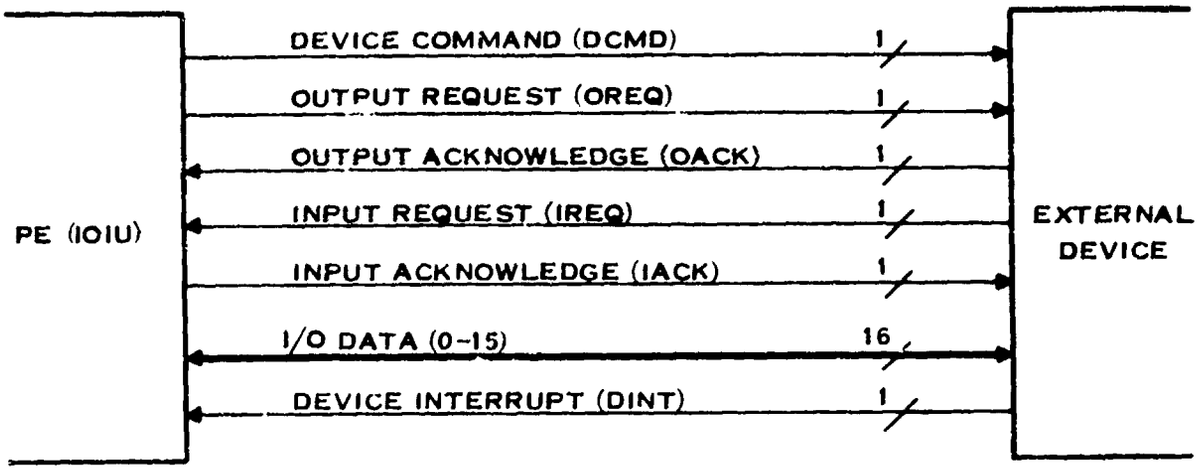
A functional register level design of the Data Transfer Channel is shown in Figure 73. The sequence of data transfers events governed by this interface is represented in the flow diagram of Figure 74. Figure 75 depicts the control word and data block structure used by the Autonomous Data Transfer Channel during channel initialization and actual data transfer procedures.

### 2. Programmable Interval Timer

A Programmable Interval Timer (PIT) was chosen as the most desirable real-time-keeping device for the DPPM application. The PIT is a down-counting register which is decremented every 50 $\mu$s. The PIT is preset with a desired time interval (in 50-$\mu$s units) and initiated under program control. The contents of the PIT can be read under program control at any time without affecting counter operation. At the completion of a count-down procedure (i.e., when the counter value reaches zero), an interrupt stimulus is generated to denote the expiration of the time interval. The PIT length is chosen at 15 bits, allowing a maximum time interval in excess of 1.6 seconds, thus offering a good compromise between timing resolution and maximum time interval capabilities.

### 3. PE Initialization Control

Immediate control of the PE subsequent to the application of power or the occurrence of a CLEAR/RESET command from an external source is manifested by "steering" the PE program execution to an appropriate memory address for each initialization event. If a system power-up




| SIGNAL | SOURCE | MEANING |
|---|---|---|
| DEVICE COMMAND (DCMD) | PE | THE I/O DATA LINES CONTAIN A VALID DEVICE COMMAND WORD. |
| OUTPUT REQUEST (OREQ) | PE | THE I/O DATA LINES CONTAIN VALID PE DATA. |
| OUTPUT ACKNOWLEDGE (OACK) | DEVICE | THE PE OUTPUT DATA HAS BEEN RECEIVED BY THE DEVICE. |
| INPUT REQUEST (IREQ) | DEVICE | THE I/O DATA LINES CONTAIN VALID DEVICE DATA. |
| INPUT ACKNOWLEDGE (IACK) | PE | THE PE HAS RECEIVED THE INPUT DATA. |
| I/O DATA (0-15) | PE/ DEVICE | IO DATA BUS (16-BITS) |
| DEVICE INTERRUPT (DINT) | DEVICE | AN ASYNCHRONOUS EVENT REQUIRING PE ATTENTION HAS OCCURRED. |

Figure 72. DP/M PE Input Output Interface Signals

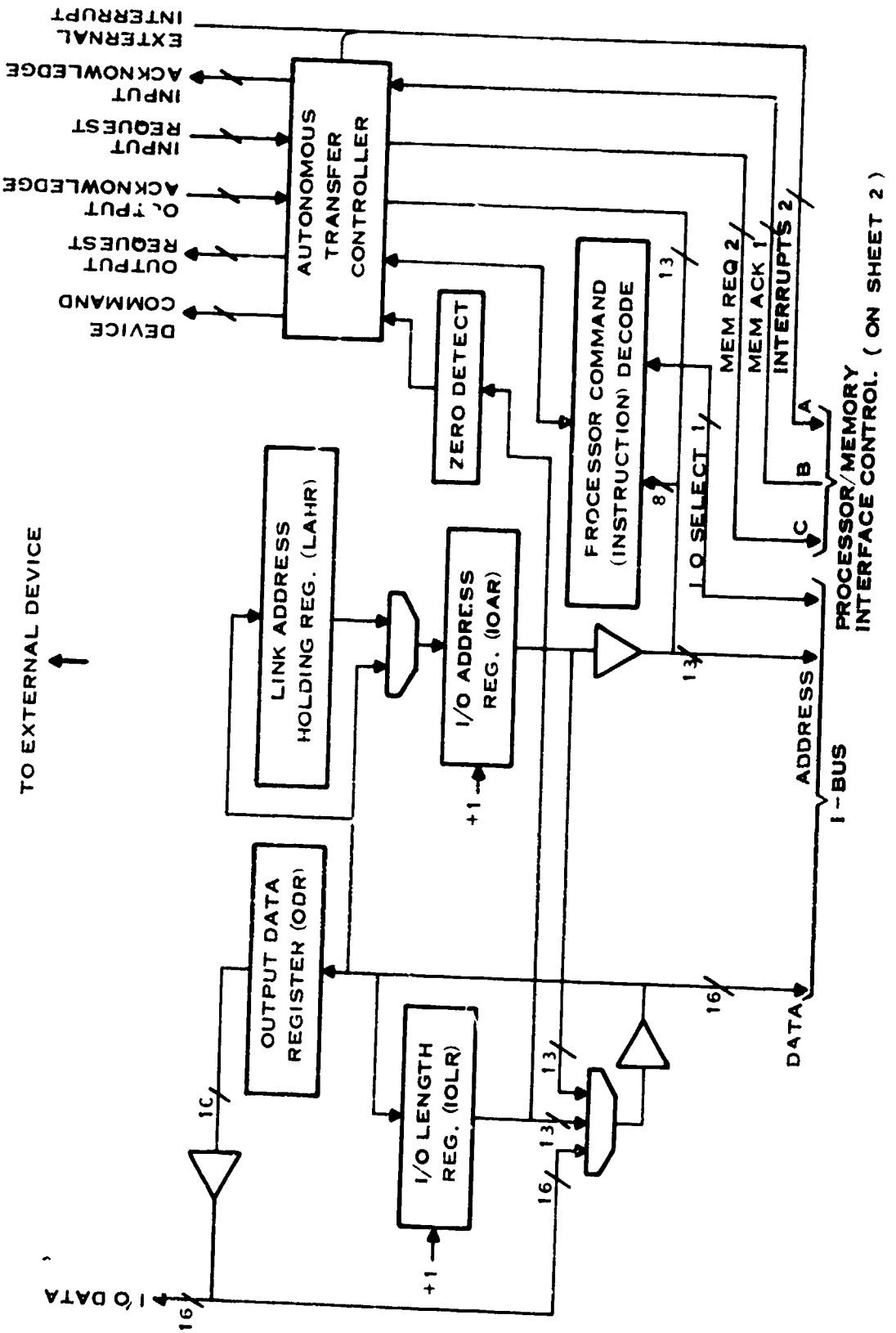



Figure 73. I/O Data Transfer Channel Register-Level Block Diagram (Sheet 1 of 2)

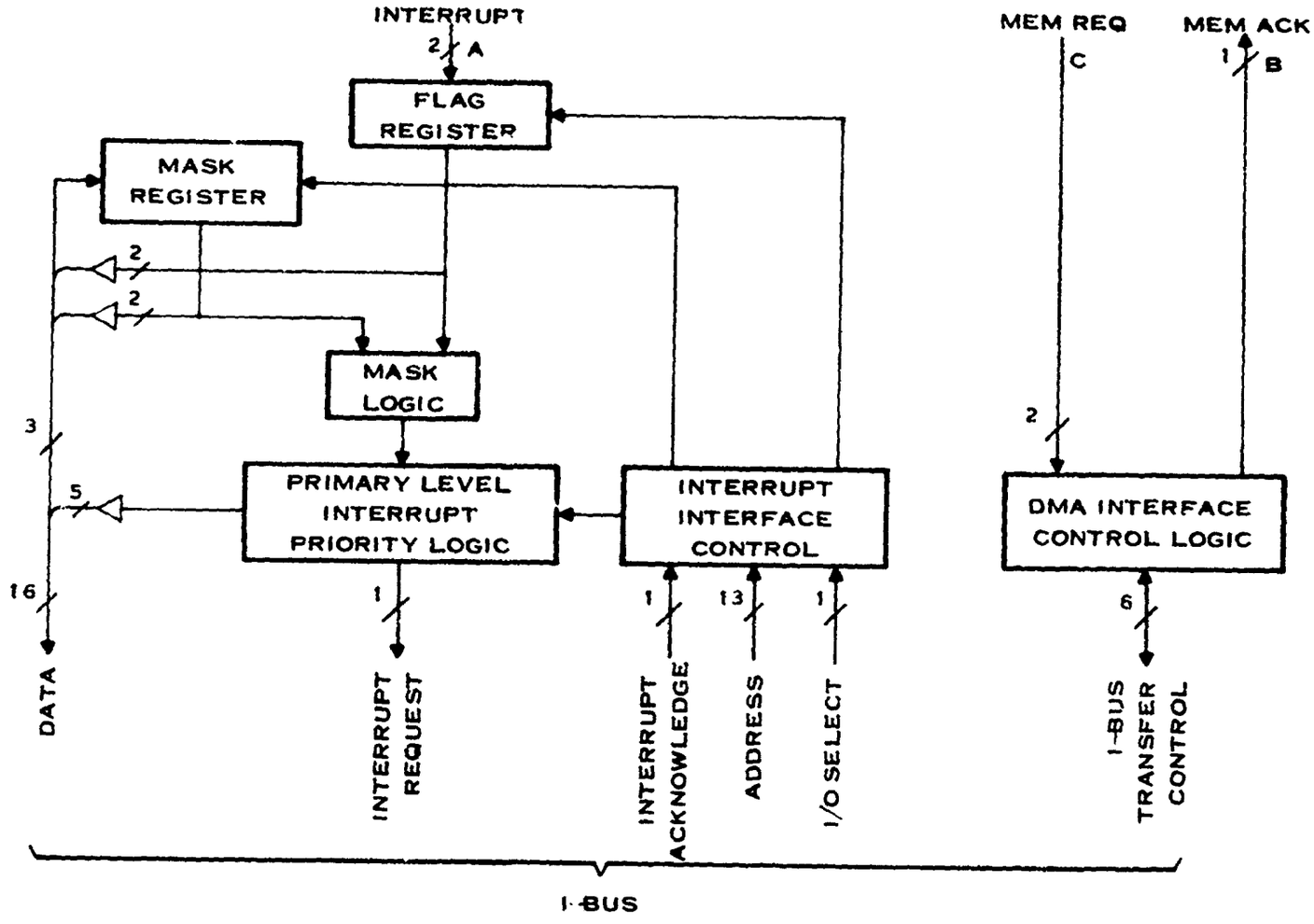



Figure 73. I/O Data Transfer Channel Register Level Block Diagram (Sheet 2 of 2)

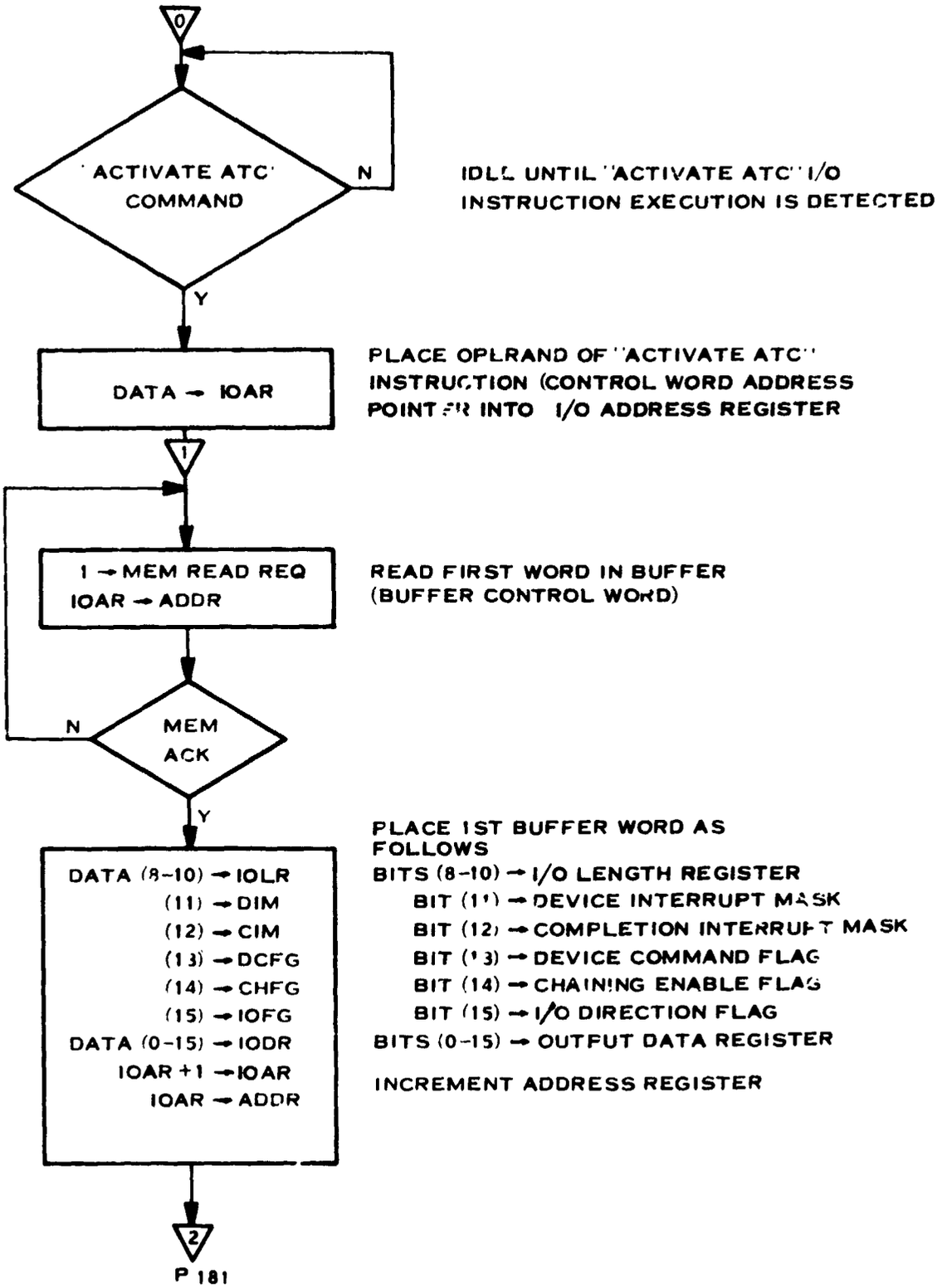



Figure 74. Autonomous I/O Data Transfer Controller Flow Diagram (Sheet 1 of 8)

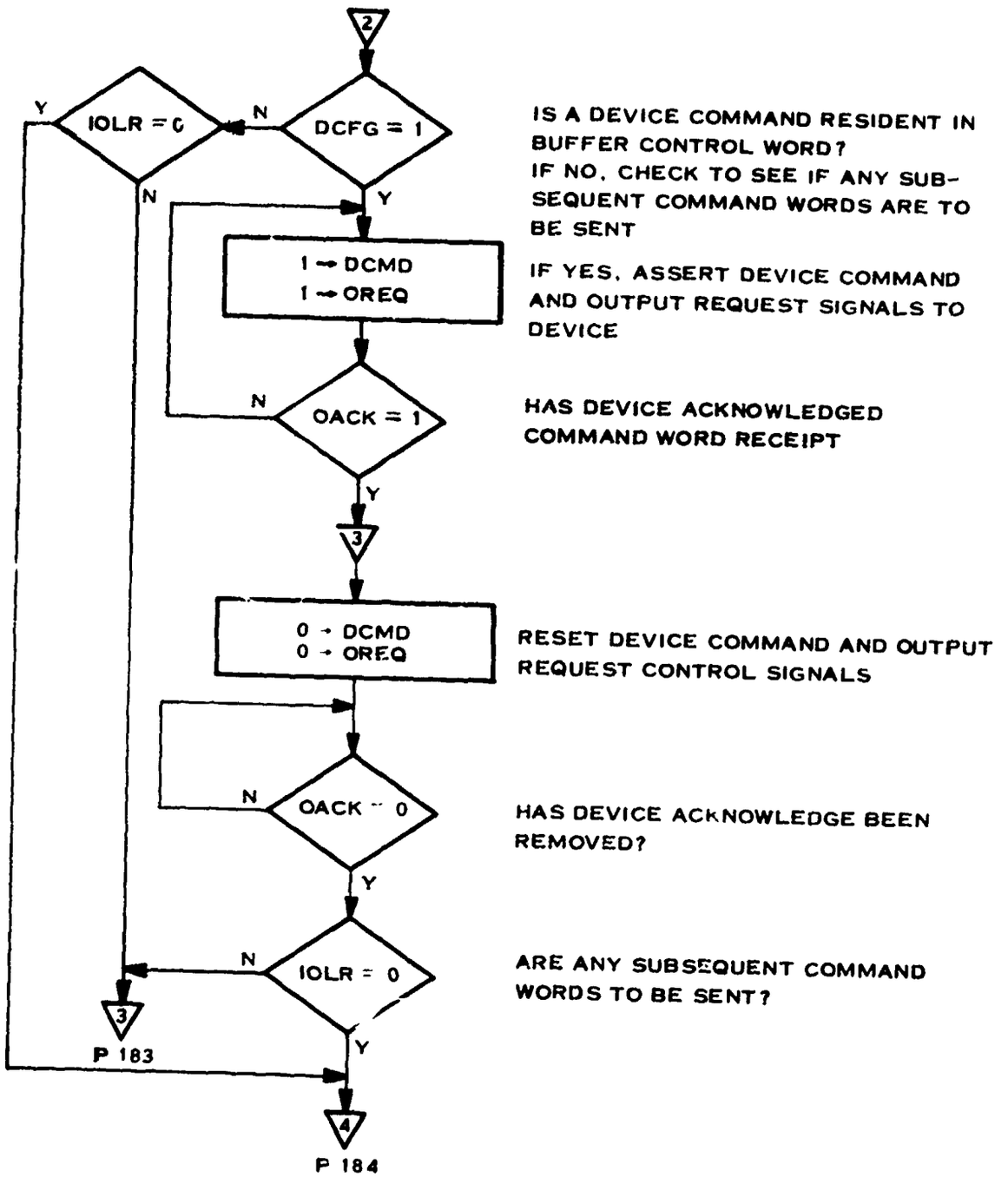



Figure 74. Autonomous I/O Data Transfer Controller Flow Diagram (Sheet 2 of 8)

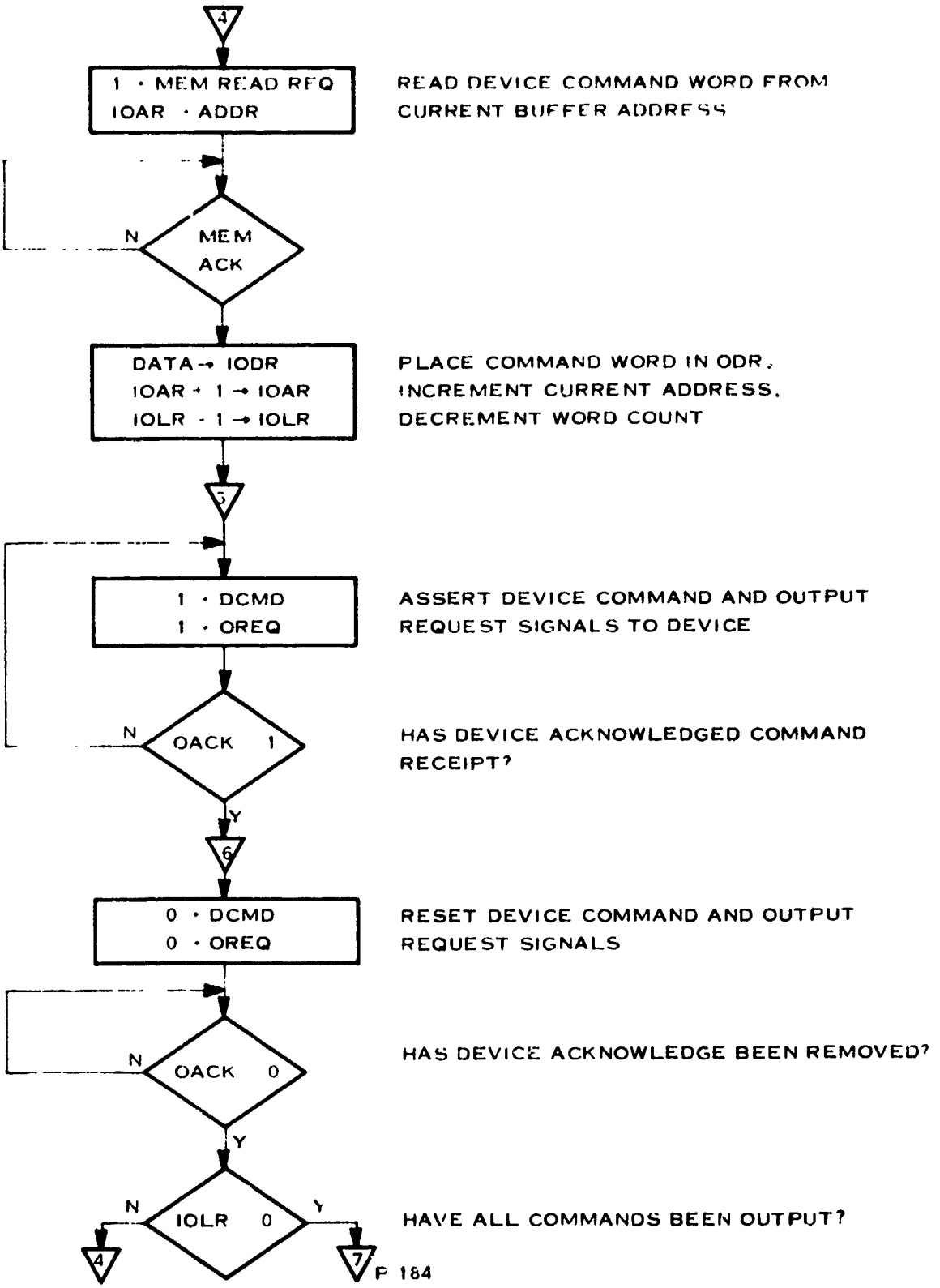



Figure 74 Autonomous I/O Data Transfer Controller Flow Diagram (Sheet 3 of 8)

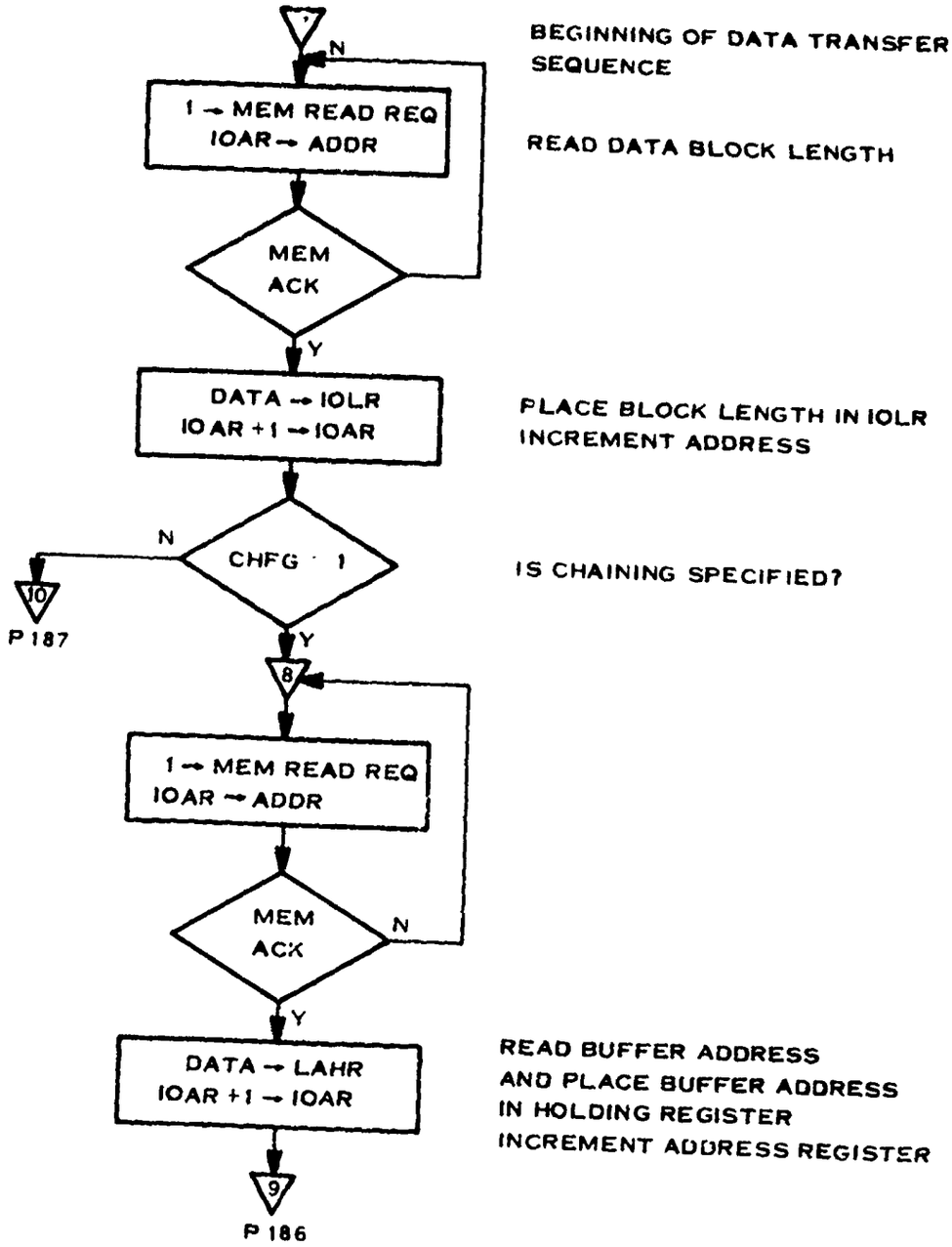



Figure 74 Autonomous I/O Data Transfer Controller Flow Diagram (Sheet 4 of 8)

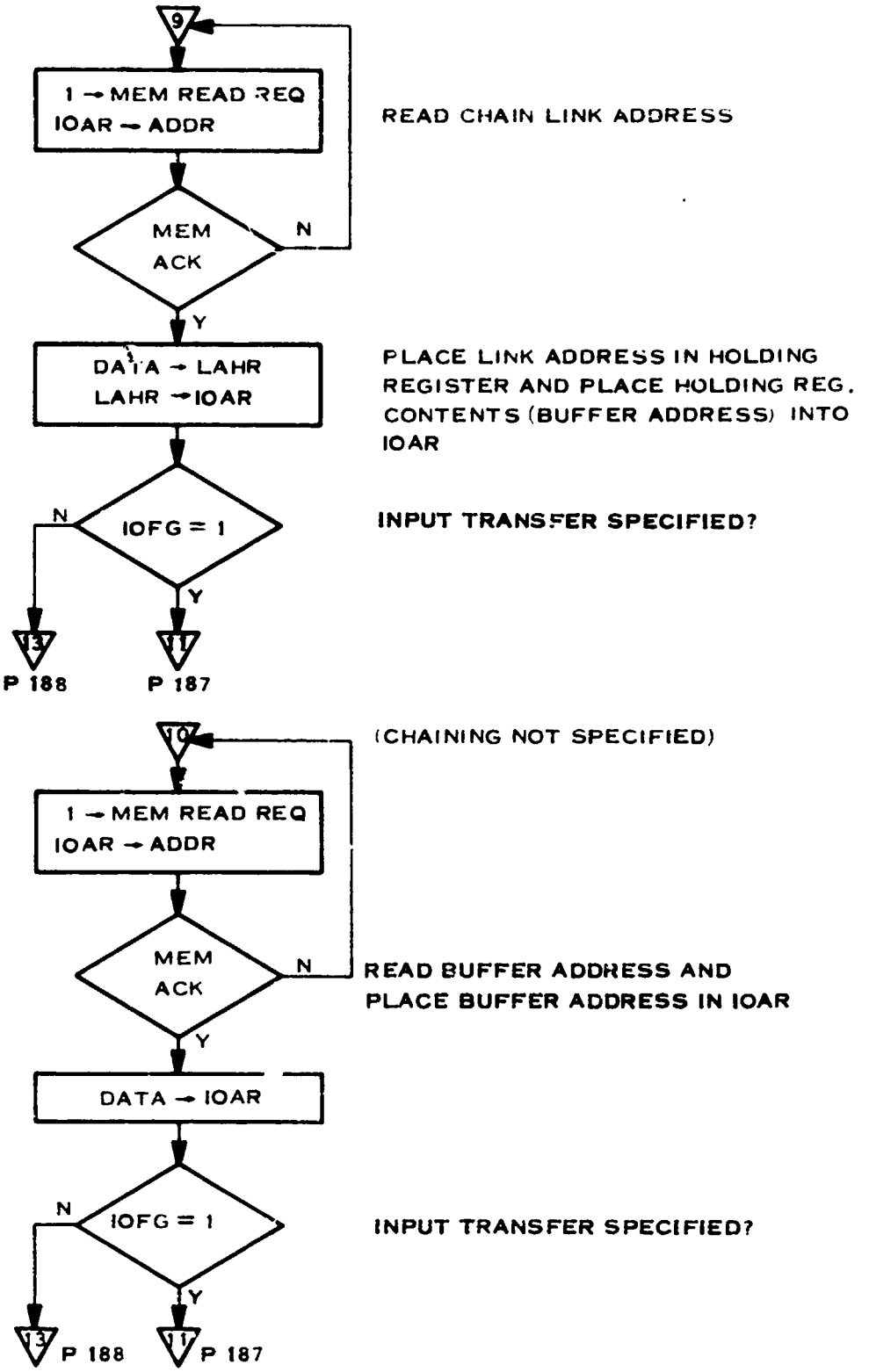



Figure 74. Autonomous I/O Data Transfer Controller Flow Diagram (Sheet 5 of 8)

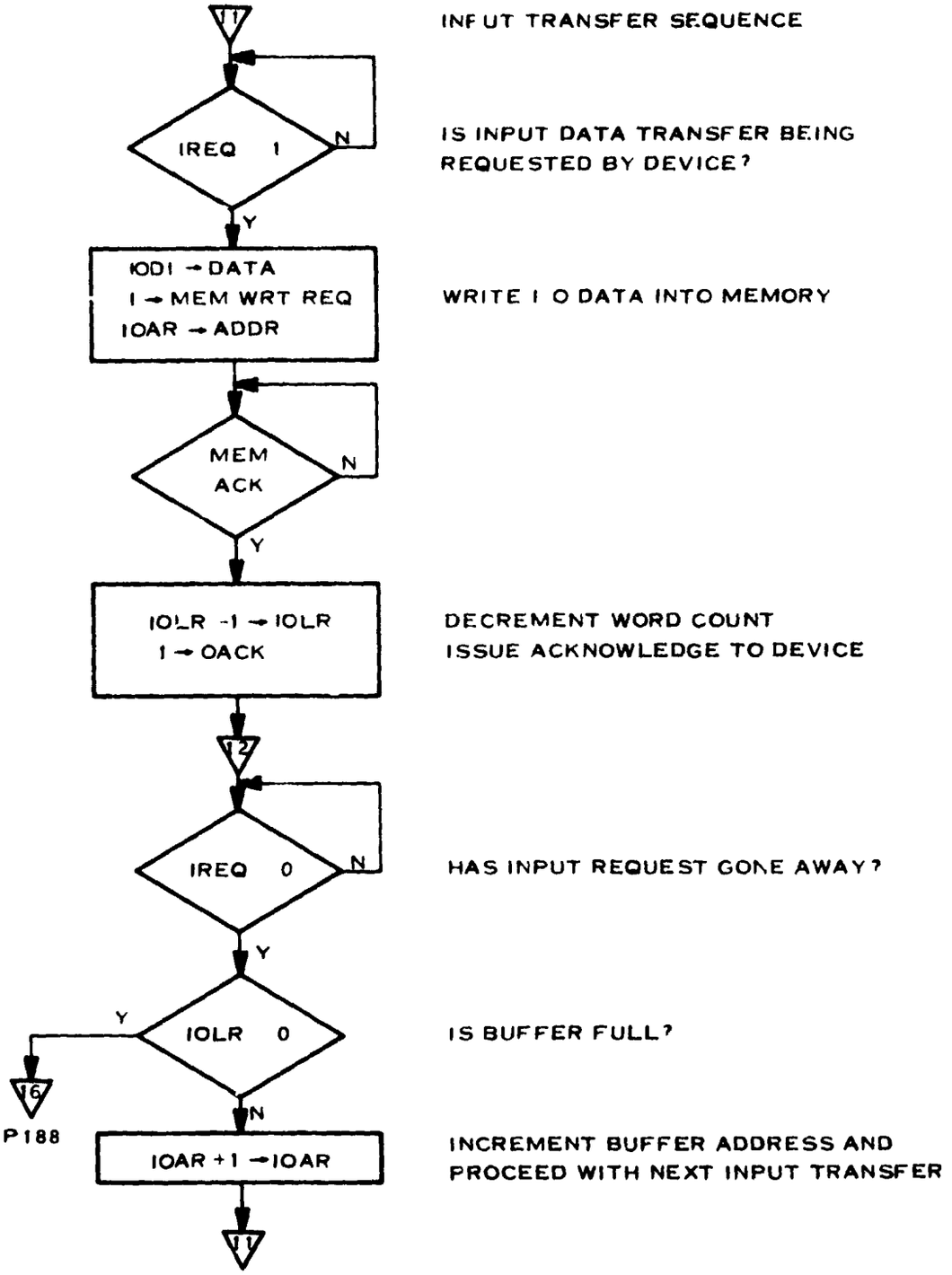



Figure 74. Autonomous I/O Data Transfer Controller Flow Diagram (Sheet 6 of 8)

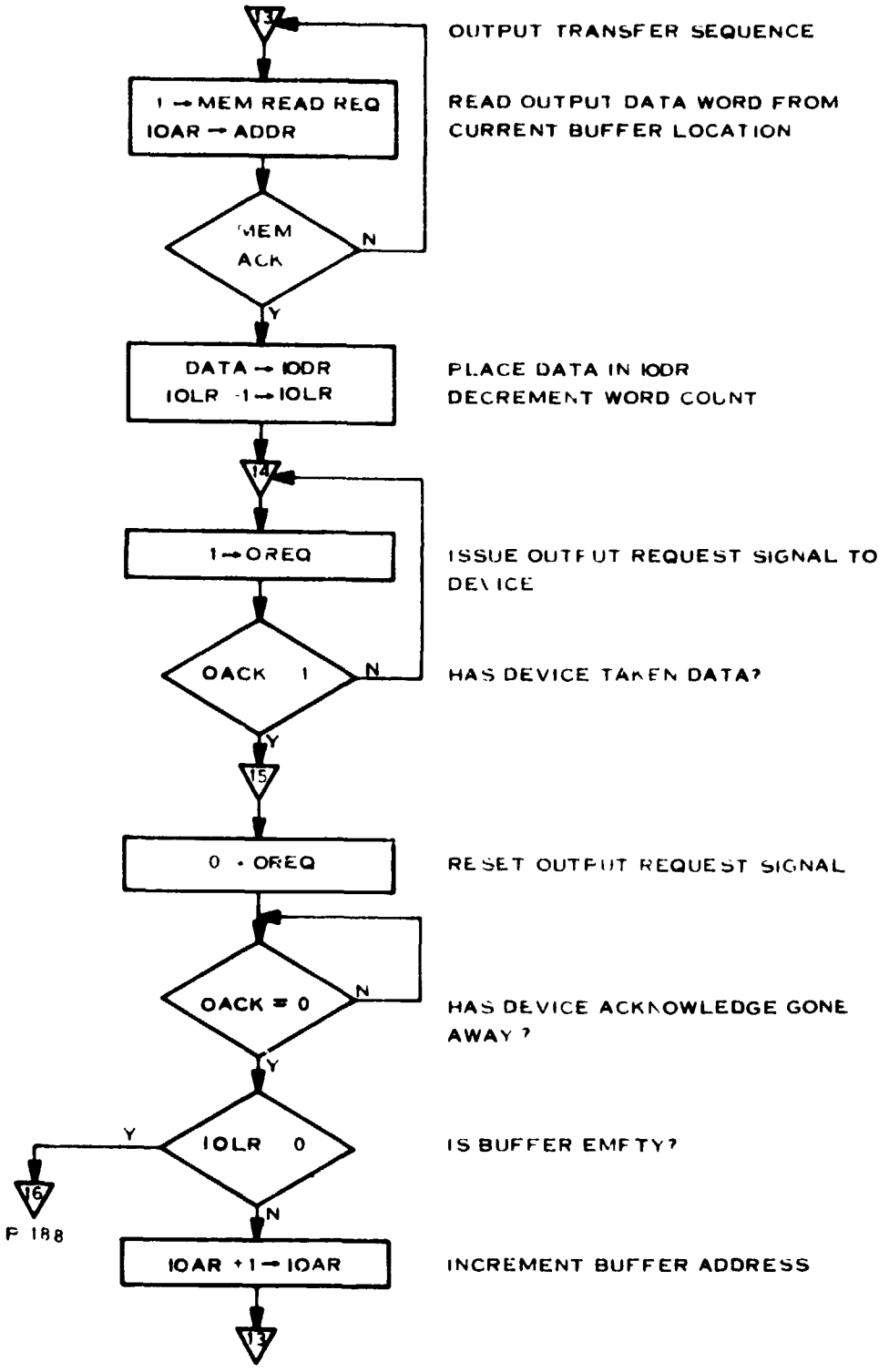



Figure 74 Autonomous I/O Data Transfer Controller Flow Diagram (Sheet 7 of 8)

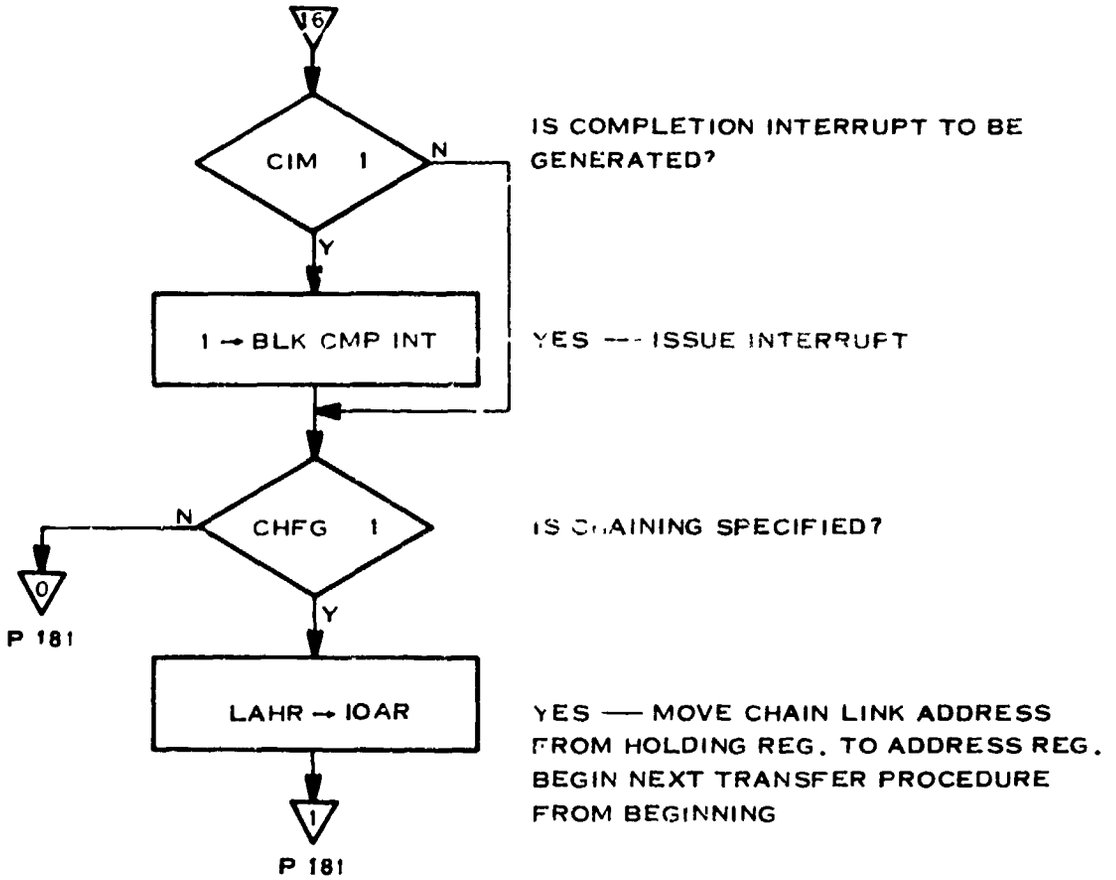



Figure 74. Autonomous I/O Data Transfer Controller Flow Diagram (Sheet 8 of 8)

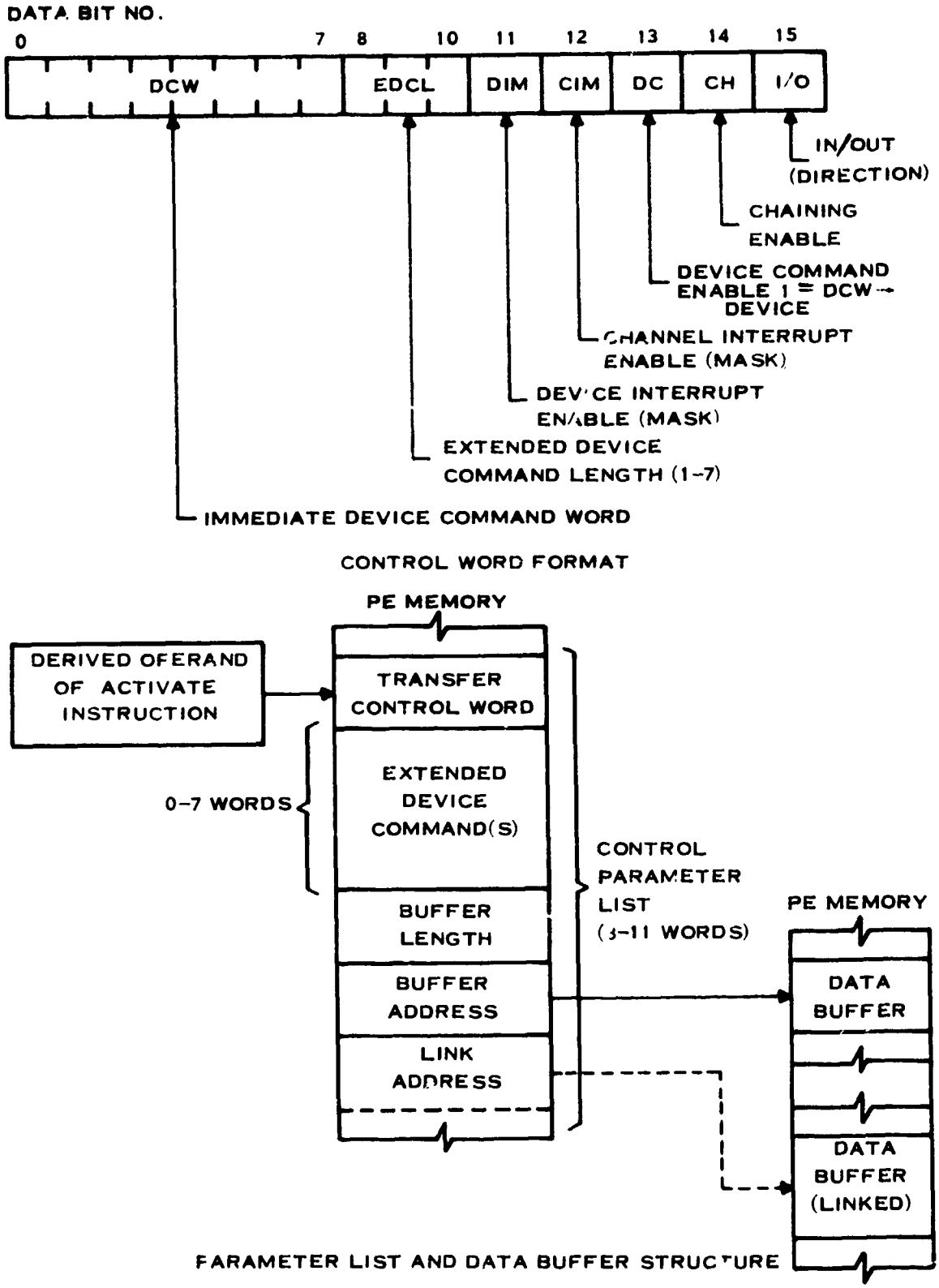



Figure 75. Autonomous Data Channel Transfer Control Word Format

or a CLEAR RESET stimulus occurs. PE initialization is performed by first providing a PE-bootstrap program starting address to the processor via the internal bus (I-BUS) interface. The PE-bootstrap program is implemented with a small Read-Only-Memory (ROM) contained in the IOIU. This memory resides in the remaining memory address space between the program/data memory limit (8K) and the maximum processor-addressable memory location (64K). It is the final responsibility of the PE-bootstrap program to enter the normal processing entry address ("GO" Address) after initialization has been accomplished.

As mentioned above, the initialization signal, CLEAR/RESET, must be externally supplied. The occurrence of this signal overrides and causes the cessation of all other PE activity, as shown in the timing diagram of Figure 76. Any I-BUS data word transfer activity presently in progress at the time of CLEAR RESET occurrence is allowed to finish (within a specified time-out period) before actual initialization activity begins to prevent possible destruction of memory data.

The Initialization Bootstrap program is a normal sequence of PE instructions contained in a small hardware Read Only Memory. The execution of this program controls all inner-PE parameter initialization as well as governing the interface between the System Program Loader device and the PE to which it is attached (via the PE's I/O data transfer channel). The actual initialization procedure followed during system start-up is discussed in Section VIII. The necessary instruction sequence required to implement this tentative initialization procedure was coded, consequently, it was decided that an ROM program size of 32 words would be satisfactory to achieve all initialization control procedures.

### 4. PE Clock Control

This functional element of the IOIU is responsible for generating the proper clock frequencies used internally to the PE and providing the necessary gating controls required to inhibit PE operation in conjunction with maintenance operations.

All internal PE clocks will be generated from an externally supplied "free" clock source. A frequency of 8 MHz is presently envisioned as the desired free clock frequency. This clock is then divided down to the appropriate PE clock frequency which is technology-dependent, but presently assumed to be in the 1- to 2-MHz range. Clock drivers are then applied to this clock signal within the IOIU (after gating) to allow distribution to all PE elements.

### 5. Processor/Memory Interface Control

This functional element of the IOIU performs the necessary control operations associated with providing interface protocol compatibility between the IOIU and the internal PE data exchange bus (I-BUS) which interconnects each functional unit of the PE. All data/command exchange between the processor/memory (P/M) and the IOIU are accomplished via this internal bus. The P/M interface control section contains the logic associated with decoding program instructions requiring IOIU action. A list of I/O commands (instructions) is given in Table 16. The P/M interface control section also contains the hardware elements required to retain and present the IOIU interrupt stimuli to the processor interrupt input via the I-BUS. This logic performs a program-controlled masking of each separate interrupt stimulus and provides protocol compliance with the processor interrupt related operations. I/O interrupts are listed in Table 17. A functional block diagram of the P/M Interface section is shown in Figure 73.

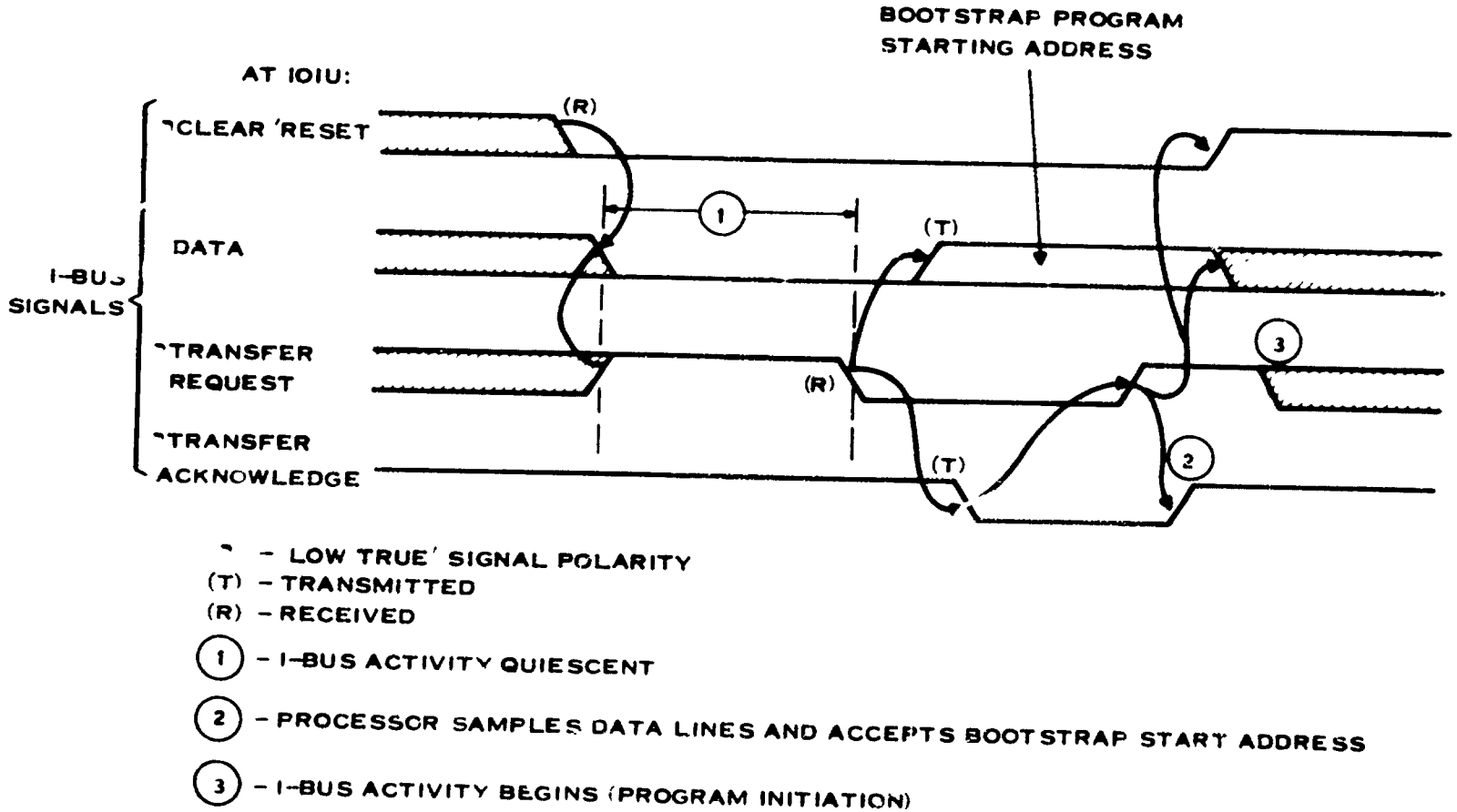



Figure 76. PE Initialization (Clear/Reset) Control Signal Sequence

**TABLE 16. IOIU COMMANDS (INSTRUCTIONS)**

**Processor Output (Write)**

1. Activate Autonomous Transfer
2. Deactivate Autonomous Transfer
3. Load Autonomous Data Channel Status
4. Load Programmable Time Interval

**Processor Input (Read):**

1. Send Autonomous Data Channel Status
2. Send Current Buffer Address
3. Send Current Buffer Length
4. Send Current Link Address
5. Send Interval Timer Value

Note Status Word Information includes

- Autonomous Data Channel Busy/Available
- Chaining Enabled/Disabled
- Current Transfer Direction (In/Out)
- Interrupt Stimulus Flags

**TABLE 17. IOIU INTERRUPTS**

1. Programmable Interval Timer
2. External Interrupt
3. Buffer Transfer Complete

Note Interrupts are listed in order of decreasing priority

## D. HARDWARE COMPLEXITY ESTIMATES

To allow a relatively accurate assessment of the impact of the above IOIU functional design on LSI implementation, a device complexity estimate was performed for each functional element of the IOIU. Complexity estimates were made in units of TTL-equivalent gates, using the same rationale and procedure as described previously in the analagous sizing effort performed for the Bus Interface Unit (BIU) in Section III. The gate sizing results are given in Table 18.

**TABLE 18. INPUT/OUTPUT INTERFACE UNIT FUNCTIONAL HARDWARE COMPLEXITY ESTIMATES**

| Functional Element | Hardware Complexity (TTL-equivalent gates) |
|---|---|
| Autonomous Data Transfer Channel | 726 |
| Programmable Interval Timer | 230 |
| PE Initialization Control | 37 + 512 bits ROM |
| PE Clock Control | 50 |
| Processor Memory Interface | 129 |

Total IOIU = 1,172 + 512 bits ROM

## SECTION VI

## PE IMPLEMENTATION CONSIDERATIONS

### A. INTRODUCTION

This section considers the technology, performance, packaging, and costs associated with DP/M PE hardware. The first part looks at the current technologies and their trends. This includes considerations of the ease of fabrication, packing density, performance, and operating temperature range. The next subsection looks at the effect of the technology, processor architecture, and the instruction execution to determine the performance that can be expected of the DP/M PE. Partitioning and packaging are examined to determine the most flexible and least expensive method of building the PE. Special consideration is given to ensure that future growth and advancements in technology can be incorporated as they become available. One feature of the partitioning that greatly enhances this ability is the separation of the processor from the memory (Figure 77) and the use of an asynchronous demand/response transfer intercommunication structure between the two modules. This allows the memory and/or the processor technology to be easily changed without adversely affecting the timing of either. Finally the cost of the PE is considered. It is shown that a DP/M-like microcomputer can offer significant cost performance features for avionics. The cost reduction for a DP/M PE is not as great as that expected for commercial microcomputers because of military environmental and reliability requirements.

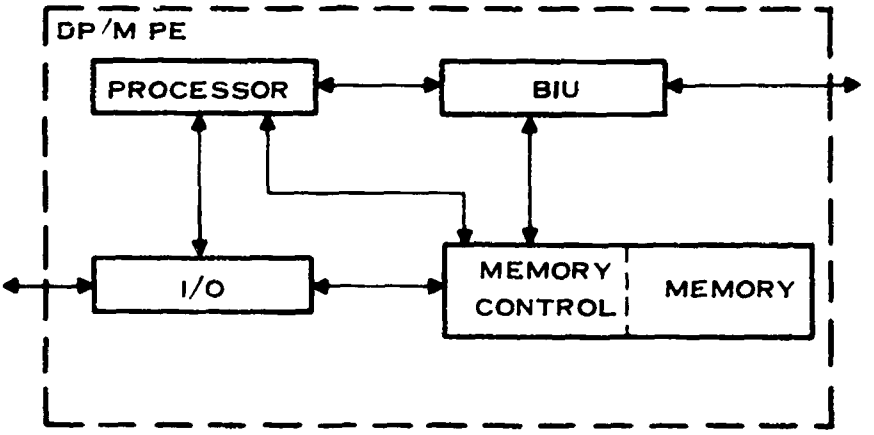



Figure 77. DP/M Block Diagram

## B. TECHNOLOGY

It appears likely that adequate DP/M PE performance can be met with current semiconductor technologies, which imply low risk and perhaps low cost. As with most systems using semiconductor devices that are currently available, the cost and performance of the system are closely tied to the technology and packaging of the system. Both current technologies and likely future technologies have been examined for their applicability to DP/M cost and performance goals.

The current semiconductor technologies include:

- Bipolar
  - Standard (TTL, DTL, ECL, etc.)
  - High density ($I^2L$, EFL, $C^2L$, etc.)
- MOS (metal oxide semiconductor)
  - PMOS (positive-channel MOS devices)
  - NMOS (negative-channel MOS devices)
  - CMOS (complementary MOS devices)
  - Special processing (silicon gate, ion implanted, depletion load, and self-aligned gate)

Future semiconductor technologies include:

- MOS
  - SOS (silicon on sapphire)
  - MNOS (metal nitride oxide semiconductor devices)
  - CCD (charge-coupled devices)
- Magnetic
  - Bubble.

The DP/M technology and packaging research has considered previous, present, and projected trends in the semiconductor industry, particularly with respect to the newer technologies. In the past, certain technologies with definite performance advantages have never been able to make the transfer from the research laboratory to the commercial marketplace. For example, silicon on sapphire (SOS) has been in the research laboratory for nearly 10 years, but has not been used extensively, except in military applications, because of its relatively high production cost when compared to that of other commercial technologies. In contrast, the integrated injection logic ($I^2L$) has gone from laboratory research in 1971 to a commercial product (as a memory cell in Intel's 256-bit RAM) introduced in June of 1973. In 1974 several new consumer (watches) and commercial (4-bit register file/ALU chip) $I^2L$ products were developed. Thus, in less than 3 years, this technology has gone from the laboratory to volume production. The DP/M technology research started with currently available technologies as candidates for baseline implementation for the PE, and the effort was directed toward the development of a cost-effective processing element for the 1978 to 1980 time period.

## 1. Memory Technology

In specifying a memory technology for DP/M, several factors were considered. All the current memory technologies, with the exception of magnetic wires, cores, and surface recording techniques, make use of photographic techniques to define the memory cell. The photographic process used has reached the level of development where it is limited by the wavelength of light. This limits line widths and or spacing to about 0.0001 inch. Thus, the size and cost of a memory cell is directly related to a cell's complexity (Table 19). The static semiconductor cells are the most complicated, typically having six devices (transistors, diodes, and or resistors) per bit cell. The newer dynamic NMOS memories have two devices (one transistor and one capacitor) per bit cell. The newer secondary memories generally have a single device per bit cell. Other factors that affect bit cell size include the device and or process complexity. The current magnetic bubble technology requires two patterned metal layers to form its cell. The dynamic NMOS requires one diffusion, two levels of conductors, and four to five mask steps, while the bipolar, CMOS, MNOS, and CCD require several more steps over the NMOS. Another factor that affects bar size, and thus cost, has to do with peripheral (on chip) circuitry. The bar utilization for memory cells ranges from a low of 22 percent for CCDs to a high of 62 percent for magnetic bubbles. These cell and bar utilization factors are what cause the 20-to-1 variation in bit density. When all these factors are taken into account, however, it is the same photographic processing that limits the bit density for all these technologies.

In selecting a memory technology for the DP/M PE, the requirement for a military temperature range has a strong influence. The high temperature makes it difficult to use any technology that makes use of stored charge such as dynamic NMOS and CCDs. The reason for this is the increased recombination and surface leakage currents associated with the +125°C operation.

Semiconductor manufacturers are gaining experience in building the current 4K dynamic NMOS memory chips. The maximum refresh time for these 0° to 70°C parts is typically guaranteed at 2 ms. The current square arrays of memory cells (64 X 64) of a 4K chip require a refresh cycle every 31 $\mu$s. With the current cycle time of 400 ns, the refresh takes about 1.3 percent of the available cycles. It has been found possible to screen parts for +90°C operation which still have 2-ms refresh times. There are indications that these parts may be able to operate at +125°C if the refresh time is reduced by 8 to 1 (0.25 ms). If designed with a 16K (128 X 128) chip operated at +125°C, it would appear conceivable that, by refreshing 128 times every 0.25 ms, the memory might be made to work. This would amount to about 11 percent of the available cycles devoted to refresh. These data indicate that it is too early to rule out a dynamic NMOS memory as a military temperature DP/M technology candidate. Conversely, other factors must be considered. There is currently no evidence that anyone has actually tried to use dynamic NMOS devices at temperature extremes in a system to determine the problems that may exist. Several people knowledgeable in the technology have serious reservations as to potential reliability problems that may exist at continued operation at elevated temperatures. Alternately, dynamic NMOS appears to be the only practical way to getting the bit density that the DP/M PE needs. All other approaches are much more expensive in terms of size, weight, and cost. Even if the dynamic NMOS device can be made to work over the temperature range, it will not be without its disadvantages.

## TABLE 19. DP/M MEMORY TECHNOLOGY

| | Bits (1024s) | Devices* per Cell | Cell Size ($mils^2$) | Bar Size (mils) | Percent | Access Time ($\mu s$) | Bit Rate (Mbps) | Write Pulsewidth ($\mu s$) | Operate Power (mW) | Standby Power (mW) | Current Operating Temperature Range (°C) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Primary | | | | | | | | | | | |
| NMOS | | | | | | | | | | | |
| Dynamic[A] | 4 | 2 | 2.3 | 156 × 184 | 32 | 0.3 | 3.0 | 0.4 | 240 | 10 | 0 to +90 |
| Static[B] | 1 | 6 | 12.1 | 126 × 164 | 60 | 0.5 | 2.0 | 0.3 | 260 | 64 | 55 to +125 |
| CMOS[C] static | 1/2 | 6 | 12.5 | 106 × 148 | 41 | 0.2 | 2.0 | 0.4 | 10 | 2 | 55 to +125 |
| Bipolar[D] static | 1 | 6 | 9.8 | 130 × 180 | 43 | 0.06 | 12.0 | 0.06 | 500 | 150 | 55 to +125 |
| Secondary | | | | | | | | | | | |
| MNOS[E] | 2 | 1 | 3.8 | 154 × 174 | 29 | 5.0 | 20.0 | 10.0 | 300 | 1 | 55 to +125 |
| CCD[F] | 1/4 | 1 | 3.1 | ≈50 × 75 | 22 | 2,500 | 0.1 | | | | 0 to +70 |
| Magnetic bubble[G] | 20 | 1 | 0.75 | ≈170 × 205 | 62 | 5,860 | 0.05 | | | 0 | 55 to +125 |

[A] Mostek

[B] AMD AM9102ADM

[C] AMI-S2222

[D] Intel

[E] Westinghouse

[F] Honeywell (experimental)

[G] Bell Laboratories (experimental)

*Active and passive devices

Because of its high refresh rate at elevated temperatures, NMOS standby power will be at least 10 times that of CMOS. Thus, any standby power source for a dynamic NMOS memory must be capable of at least 10 times the power that a CMOS memory would require to be nonvolatile. Of course, the advantage of the dynamic NMOS is 4 times the bit density. For the DP/M baseline the primary choice is dynamic NMOS for packing density followed by CMOS for its better temperature and power characteristics. A possible contender is Integrated Injection Logic ($I^2L$). The $I^2L$ technology seems to have good potential but has not seen extensive memory application to date. Many of the read mostly memories (RMM) such as MNOS are either quite slow by comparison and or much more difficult to produce. Because of these reasons there appears to be little commercial interest in such technologies, and high-volume commercial production is one means to achieve the desired DP/M cost goals.

The recommended baseline DP/M memory chip has the following features

*16K bits/chip (1978 to 1980)

8K words × 2 bits

Single device cell dynamic NMOS

0.33 to 0.5 μs access time.

Over the next 5 years, substantial improvements can be expected with representative characteristics shown in Table 20.

*a. Dynamic NMOS*

With improved layout and sense amplifier design, a 2-to-1 improvement in cell size coupled with a larger bar should allow dynamic NMOS to reach 16K bits per chip.

*b. Static NMOS*

A 2-to-1 improvement in layout and bit cell, coupled with a 2-to-1 bar size increase, should allow static NMOS to reach 4K bits per chip.

*c. CMOS*

With improved layout and isolation techniques (such as SOS), a 4-to-1 improvement in cell size coupled with a larger bar should allow CMOS to reach 4K bits per chip.

*d. Bipolar*

With improved isolation techniques (such as oxide isolation) and improved memory cells (using integrated injection logic), a 4-to-1 improvement in cell size should allow bipolar memories to reach 4K bits per chip.

*e MNOS*

With improved layout and device technology, a 4-to-1 improvement in cell size and an increased bar size should allow MNOS memories to reach 16K bits per chip.

## TABLE 20. CURRENT MEMORY DEVELOPMENT

| | Bits (1024s) | Devices* per Cell | Cell Size ($mils^2$) | Bar Size (mils) | Percent | Access Time ($\mu s$) | Bit Rate (Mbps) | Write Pulsewidth ($\mu s$) | Operate Power (mW) | Standby Power (mW) |
|---|---|---|---|---|---|---|---|---|---|---|
| Primary | | | | | | | | | | |
| NMOS | | | | | | | | | | |
| Dynamic[A] | 16 | 2 | 1.0 | 200 × 200 | 40 | 0.2 | 4 | 0.3 | 350 | 20 |
| Static[B] | 4 | 6 | 6.0 | 200 × 200 | 60 | 0.4 | 2.5 | 0.4 | 300 | 100 |
| CMOS[C] | | | | | | | | | | |
| Static | 4 | 6 | 5.0 | 200 × 200 | 50 | 0.3 | 2 | 0.5 | 10 | 2 |
| Bipolar[D] | | | | | | | | | | |
| Static | 4 | 6 | 3.1 | 160 × 150 | 50 | 0.05 | 20 | 0.05 | 200 | 20 |
| Secondary | | | | | | | | | | |
| MNOS[E] | 16 | 1 | 1.0 | 200 × 200 | 40 | 2.0 | 5 | 5.0 | 300 | 1 |
| CCD[F] | 32 | 1 | 0.34 | 154 × 168 | 43 | 500 | 0.5 | | 320 | 20 |
| Magnetic bubble[G] | 256 | 1 | 0.1 | 200 × 200 | 60 | 600 | 0.5 | | | 0 |

[A] Stein-Siemens

[B] Projected from present

[C] Projected from present

[D] Wiedmann IBM

[E] Projected from present

[F] Agusta-IBM

[G] Projected from present

*Active and passive devices

*f CCD*

With improved layout and multiple bits per cell, a 10-to-1 improvement in cell size, a 2-to-1 improvement in bar utilization, and a larger bar should allow CCD memories to reach 32K bits per chip.

***g. Magnetic Bubble***

By developing the single patterned metal layer technology coupled with X-ray photographic processing (reduced wavelength to allow smaller line width), an 8-to-1 improvement in cell size coupled with a larger chip should allow magnetic bubbles to reach ~56K bits per chip.

**2. Logic Technology**

For the processor, BIU, and I/O logic chips, several LSI technologies have the required performance of under 4 μs average instruction time (Figure 78). These include NMOS, SOS-CMOS, and $I^2L$.

As with the memory, one of the most important considerations in selecting an LSI technology for the processor is the operating temperature range and the manufacturing associated with the technology cost. The bipolar technology (e.g., TTL, $I^2L$) has a major advantage over the MOS technologies in that it does not require special layouts and or processing to operate over a full military temperature range. To operate over this range, MOS devices generally require either a much larger layout (2 to 1) with guard rings around each device (as with CMOS, Figure 79) or special processing as with SOS. These special techniques cause the military version to be significantly more expensive than a commercial version. Thus, from a cost standpoint, it would be better to use a bipolar technology such as $I^2L$ for the processor and take advantage of any commercial production base. An additional advantage of bipolar is the ease of interfacing, which could significantly affect the overall system cost and flexibility.

The best bipolar LSI candidate is Integrated Injection Logic ($I^2L$) which applies linear circuit techniques to form a high-density, low-power logic circuit. From a circuit standpoint, a lateral PNP transistor is used as a current source load for the inverted NPN logic transistor (Figure 80). The logic transistor can have multiple collector outputs, and the desired logic operation is performed by wire-ANDing several of these collectors together. The $I^2L$ approach uses new and novel device and circuit techniques rather than a new technology to achieve its high performance and packing density. Of course, as with any device series, some process optimization will be required to achieve high yields and improve performance.

In addition to operating temperature range, the other important items to consider in the selection of an LSI technology are power dissipation (speed power), performance (propagation time), and chip size (gates per unit area). Of the available technologies (Table 21), $I^2L$ has the best speed, power, and gate density attributes. It is faster than MOS devices, but somewhat slower than other bipolar devices. Further, as pointed out above, since it is a bipolar device, a military temperature range version can be simply selected from the commercial devices, thus taking advantage of volume commercial production. $I^2L$ is the recommended technology for the logic portion of DP/M. There should be no problem interfacing $I^2L$ logic with NMOS, CMOS, or

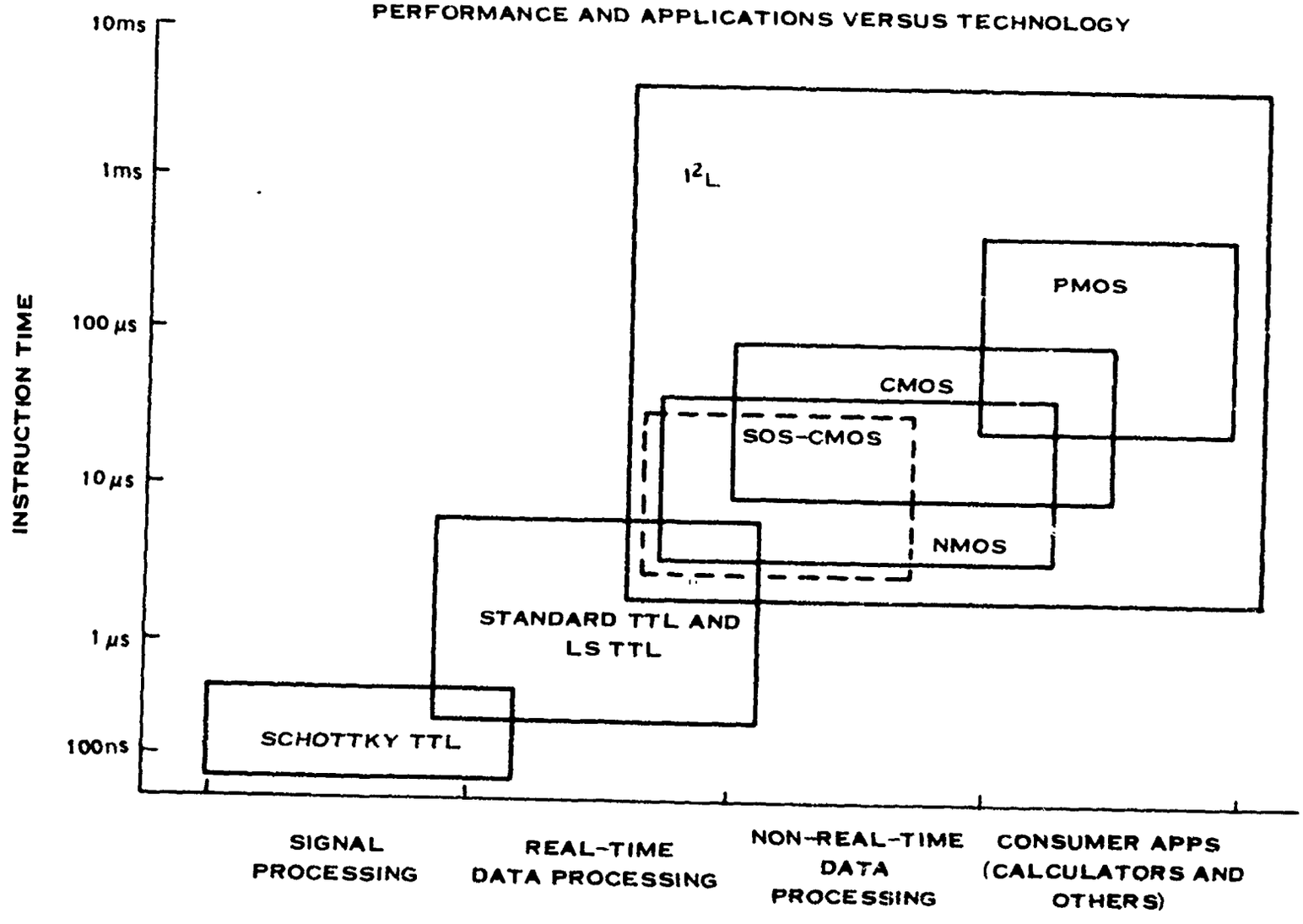



Figure 78. Performance and Applications vs Technology

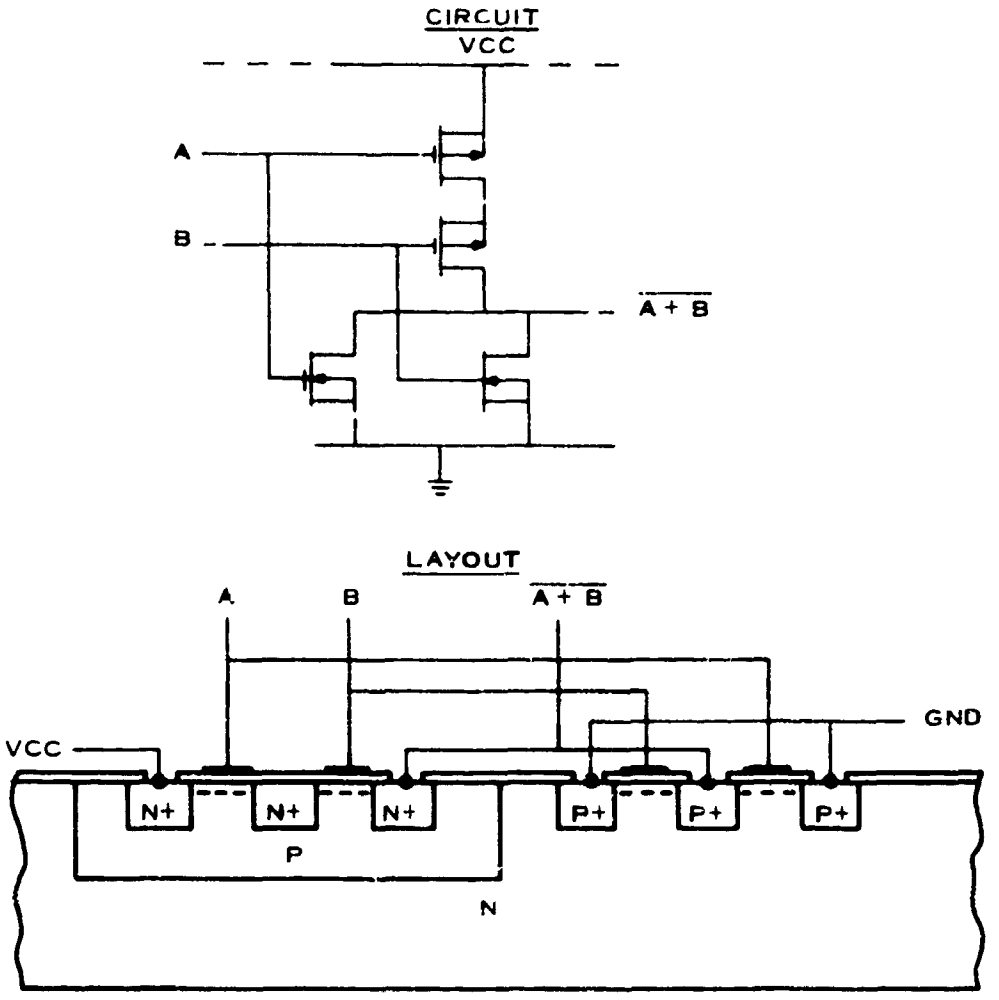



Figure 79 Complementary MOS (CMOS)

TABLE 21 DP/M PROCESSOR TECHNOLOGY

| | Speed Power (pJ) | Propagation Time (ns) | Gates* per 10,000 mils$^2$ |
|---|---|---|---|
| Standard bipolar (TTL, DTL, ECL) | 10 to 100 | 2 to 20 | 70 to 200 |
| High density bipolar ($I^2L$) | 1 to 5 | 20 to 100 | 400 to 700 |
| MOS (PMOS, NMOS, CMOS) | 10 to 100 | 30 to 300 | 200 to 500 |

*Equivalent gates include leads, bonding pads, and bar edges

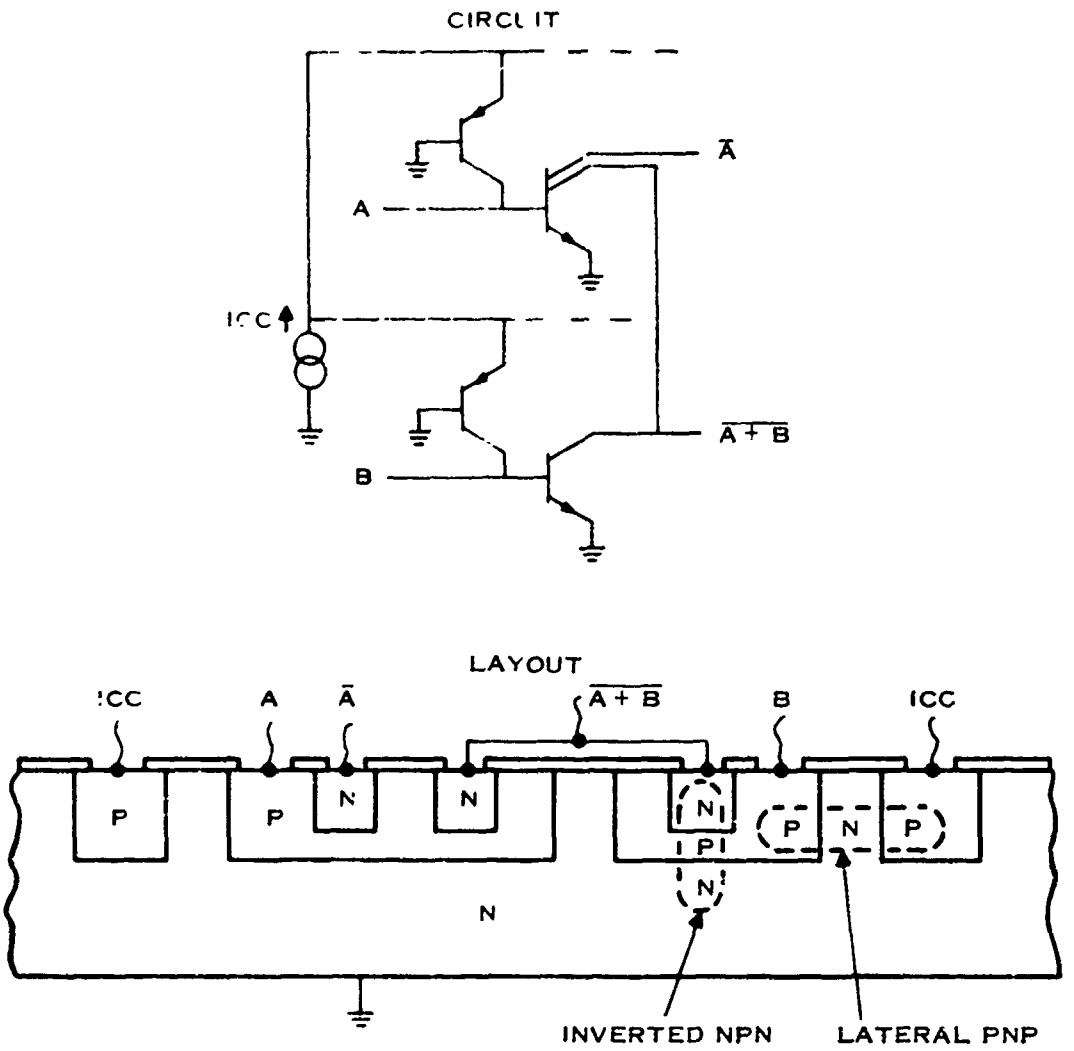



**Figure 80. Integrated Injection Logic $I^2L$**

bipolar memories since these technologies are all normally built with TTL level interface compatibility. The recommended DP/M baseline logic chips have the following characteristics:

$I^2L$ technology

3K to 4K gates chip (1978 to 1980)

80 $\mu$W and 20 ns internal gate

2 mW input (TTL compatible)

6 mW output (TTL compatible)

It should be noted that some of the more complex $I^2L$ devices currently in production (e.g., 4-bit processing element chips) have up to 1,450 gates per chip. Thus, with the partitionings

shown in Section IV (especially the 2-chip processor) it is possible to start development of the DP/M processor chip(s) in 1975. The most likely alternative to $I^2L$ is NMOS which has good density and performance. Its major disadvantage is that it does not normally perform well over the the full military temperature range. Neither CMOS nor SOS technologies appear very attractive because of their relatively expensive manufacturing process and lack of commercial production base experience. Both technologies have advantages for military applications but neither have done well in the commercial high-volume low-cost market.

### 3. Summary of Technology Recommendations

Based on the current state of technology development, it appears that dynamic NMOS is the best choice for the memory if it can be made to work over the military temperature environment. The best alternative is probably CMOS with its lower power and better temperature characteristics, but at the expense of density. For the logic portion of DP/M the newer $I^2L$ technology appears to be the best choice. One potential advantage of $I^2L$ is that it may be easier to make it radiation hardened without adversely affecting its cost, based upon the following general information:

- $I^2L$ is a bipolar semiconductor technology and is therefore comparatively "harder" than MOS technology
- $I^2L$ has few, if any, nonactive device functions to be exposed to radiation
- Photocurrent (i.e., radiation-induced current) is one of the methods used to power an $I^2L$ device.

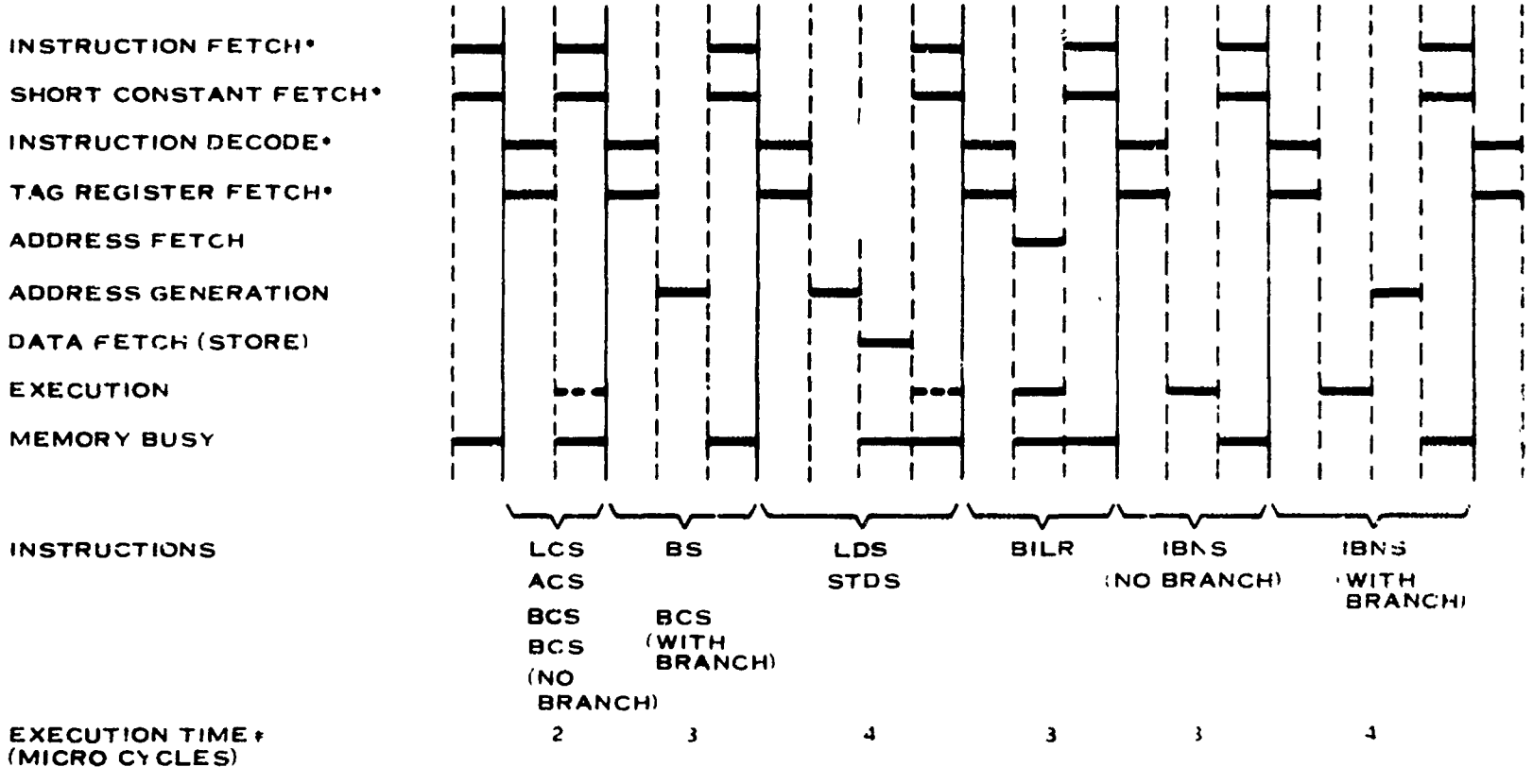
The best alternative is probably NMOS except for its potential temperature problem. The next choice would be CMOS, but it suffers from poor packing density.

## C. PE PERFORMANCE PROJECTIONS

The performance of the DP/M processor is dependent on the number of microsteps required for each operation and the time for each microstep, plus the memory speed. Examples of the number of microsteps required for instruction execution are given in Section IV. The number ranges from as few as two for the simpler instructions, such as Add Constant Short and Add Register, to over 20 for multiply and divide. The time for each microstep depends on the number of levels of logic (gates) between registers and the propagation time per level. Normally, the number of levels of logic between registers is dominated by the control and carry propagation paths through the arithmetic logic unit (ALU). This path, counting the multiplexers, ALU data/control/carry, and register input/output, typically has 10 to 20 levels of logic. Depending on the logic technology used, the propagation time per level can be in the 20- to 50-ns range. This gives a range of 0.2 to 1.0 $\mu$s per processor microcycle. By minimizing the number of logic levels and projecting technology performance, it would appear that microcycles in the 0.25- to 0.5-$\mu$s range in the 1978 to 1980 time period are reasonable for the DP/M PE.

When considering the memory performance needed to match the processor, the number of levels of logic in the "address out" and the "data in" lines should be minimized. Then to match the memory to the processor, the memory should have an access time 20 to 30 percent less than the processor's microcycle, and the memory cycle should be no longer than the processor microcycle. The memory technologies that have been proposed should have access and cycle times with this range in the 1978 to 1980 time period. Thus, the DP/M PE should be able to run with a 2- to 4-MHz clock. Figures 81 and 82 show the microcycles involved in the extended

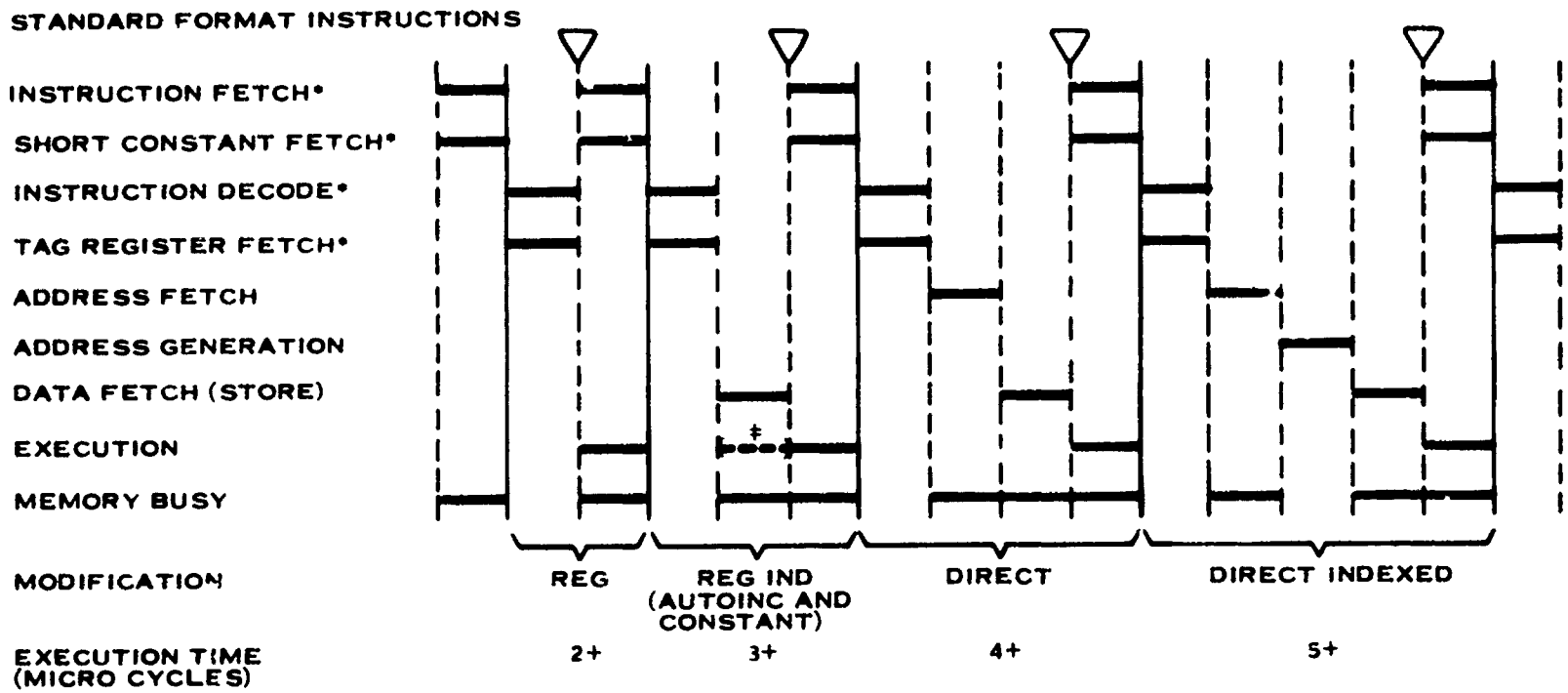
EXTENDED SHORT INSTRUCTIONS




* ALWAYS PERFORMED INDEPENDENT OF NEXT INSTRUCTION.

* ASSUMES MEMORY CYCLE = PROCESSOR CYCLE AND MEMORY ACCESS = 3/4 PROCESSOR CYCLE.

Figure 81. Extended Short Instructions




▽ EXTENDED EXECUTION TIME BY THE FOLLOWING MICRO CYCLES

| | | | |
|---|---|---|---|
| SFTS | 2+N | XSPC AND RINT | 6 |
| SFTD | 2+N | PSHM AND POPM | N |
| M AND DV | 20 | BRANCH | 1 |

‡ EXECUTION (INCREMENT REGISTER) FOR REGISTER INDIRECT AUTOINCREMENT AND CONSTANT MODIFICATIONS.

• ALWAYS PERFORMED INDEPENDENT OF NEXT INSTRUCTIONS.

Figure 82. Standard Format Instructions

short and standard short format instructions. With the exception of a few instructions such as multiply, divide, and shift, the instructions take from 2 to 5 microcycles. Excluding these instructions and assuming an average of 4 microcycles per instruction would give a processing rate of 500 to 1000 kilo instructions per second (KIPS). Including 10 percent multiply/divide operations would drop the rate to 330 to 660 KIPS. This level of DP/M PE performance will be more than adequate to do a large portion of the avionics problem.

## D. PE PARTITIONING AND PACKAGING

Comparing the complexity of the various PE elements based on the DP/M design analysis (Table 22) to projected semiconductor technology trends, it appears that in 1978 to 1980 the PE cannot be economically fabricated as a single semiconductor chip. Further, the memory complexity dominates the PE. An important consideration in the partitioning of the PE is to make it modular to be able to take advantage of technology advancements and commercially available parts and tailor the processing element memory and I/O to the particular application.

There are several factors to be considered in specifying the partitioning and packaging for DP/M PE. These include size, weight, cost, and reliability. The current semiconductor industry standard is the dual in-line package (DIP). This package is rectangular in shape and has two rows of pins on 100-mil centers pointing down along the long sides. Low-cost plastic and high-reliability ceramic packages are commercially available in this form factor. There are currently packages with 6, 8, 14, 16, 18, 20, 22, 24, 28, 40, 48, and 64 pins with this form factor. The ceramic version of this package is the one recommended for LSI devices for the DP/M PE.

**TABLE 22 PROCESSING ELEMENT COMPLEXITY**

| Functional Unit | Complexity (gates/bit) (X1000) | Packages | | Pins/Package | Power (watts) |
|---|---|---|---|---|---|
| | | Type | Total | | |
| Processor | 3 to 4 | 1 to 2 | 1 to 2 | 48 | 0.7 |
| Memory controller | 0.6 to 0.8 | 1 | 1 | 48 | 0.2 |
| Memory | [c]128 to 132 | 1 | 8 to 9 | 20 | 3.2 |
| Bus interface[a] | 4 to 6 | 3 | 8 | 14, 20, 48 | 1.7 |
| I/O interface[b] | 1 to 1.5 | 2 to 6 | 6 to 9 | 16, 18, 24, 28, 40, 48 | 1.9 to 2.1 |

[a] Estimates do not close bus line driver/receiver devices

[b] Variation in estimates is caused by different partitioning approaches. Device quantity estimates do not include a few miscellaneous standard devices required for PE initialization control and PE clock control

[c] Without and with 1-bit parity

Note: Power dissipations shown are conservative, worst case estimates reflecting implementation with both $I^2L$-LSI (low-power) devices and standard technology (low-power Schottky bipolar) devices

The next consideration is whether the whole PE should be placed into a single hybrid package using beam lead (or wire-bonded) chips or into multiple packages mounted on a printed circuit board. The single hybrid package has the smallest size and weight and also has the highest reliability if beam lead chips are used. Its major disadvantage is a very high cost. The hybrid package can be as much as 10 times as expensive as separate packages mounted on a printed circuit board. When the size of all the peripheral devices such as clocks, power supplies, and interface circuitry are considered, it is unlikely that the modest size reduction gained by using the hybrid package can justify its high cost. Thus, for the DP/M PE, it is recommended that each integrated circuit chip be mounted in a separate package. The PE is then fabricated by placing these packages onto a printed circuit board using conventional techniques. This should result in a PE with an approximate 4- by 5-inch (excluding peripheral circuitry) form factor. Because of the low power and small size of the DP/M PE, it is envisioned that the PE will normally be a part of some larger system. As such, it is expected that the PE would be a single card within a line replaceable unit (LRU). In this case it would share power, cooling, and packaging with the remainder of the equipment within the LRU.

The number of pins per package also affects the package size and greatly affects the device cost. The partitioning of the PE must be done carefully to minimize the number of pins per package. Conversely, as the number of pins is reduced beyond some point, performance and chip count will suffer because different information must be multiplexed onto the same lines. Examples of this are the processor's data and address sharing the same lines. The address is placed on the lines first and latched into an external register (possibly on the memory chips). Then the data is transmitted over the same set of lines. The disadvantage is the increase in execution time, external circuitry, and complexity. The advantage is a 2- to 1 reduction in address/data pin count. As discussed in Section IV, it is recommended that the data and address be placed on different pins to maintain maximum performance.

### 1. Processor

One example of a commercially available 8-bit microprocessor is the Intel 8080 which uses a 40-pin DIP. The 40 pins are divided into 8 data, 16 address, and various control, clock, and power pins. The next larger commercially available package is a 48-pin DIP that will be adequate for the DP/M processor. This would allow for 16-bit data, 16-bit address, and the various control, clock, and power pins. Figure 83 shows a one- and a two-chip partitioning of the processor using this 48 pin package. Details of the processor design and complexity are given in Section IV.

### 2. Memory and Control

As discussed in Section IV, the memory control is a separate DP/M element (Figure 84). This controller handles memory refresh, parity, and write protect of the memory chip shown in Figure 85. In addition, the memory control chip handles the I-BUS timeout watchdog timer mentioned in Section IV. The refresh (if required) is controlled by a memory control chip which allows easy modification to handle different classes of memories. A bus address decoder is also included to differentiate between I/O and memory operations. If other than the first 8K of memory is to be addressed by a controller, an inverter must be placed in one or more of the memory control address lines. The memory controller has a chip-enable line which drives the nine 16K-bit memory chips.

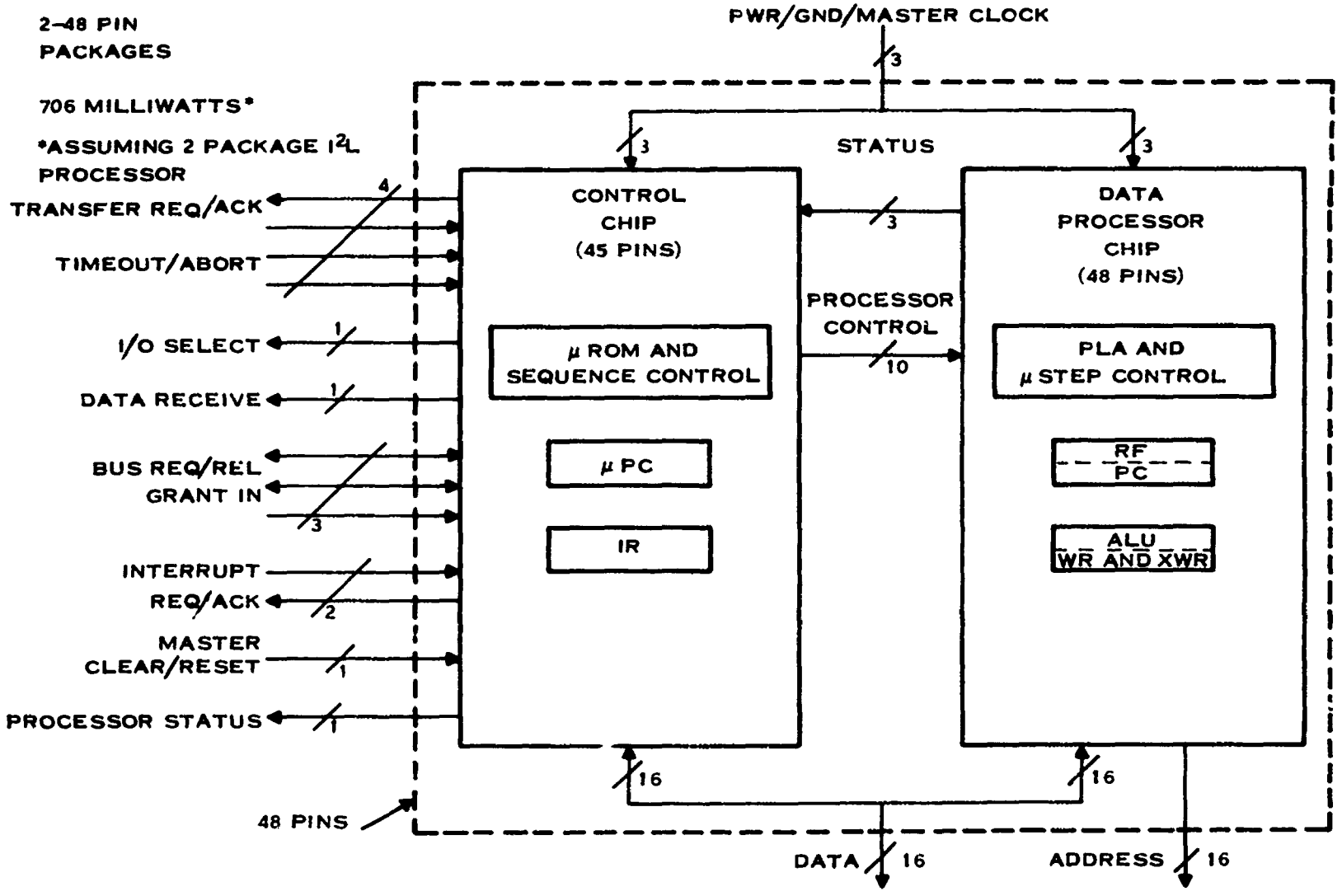



Figure 83. DP/M Processor

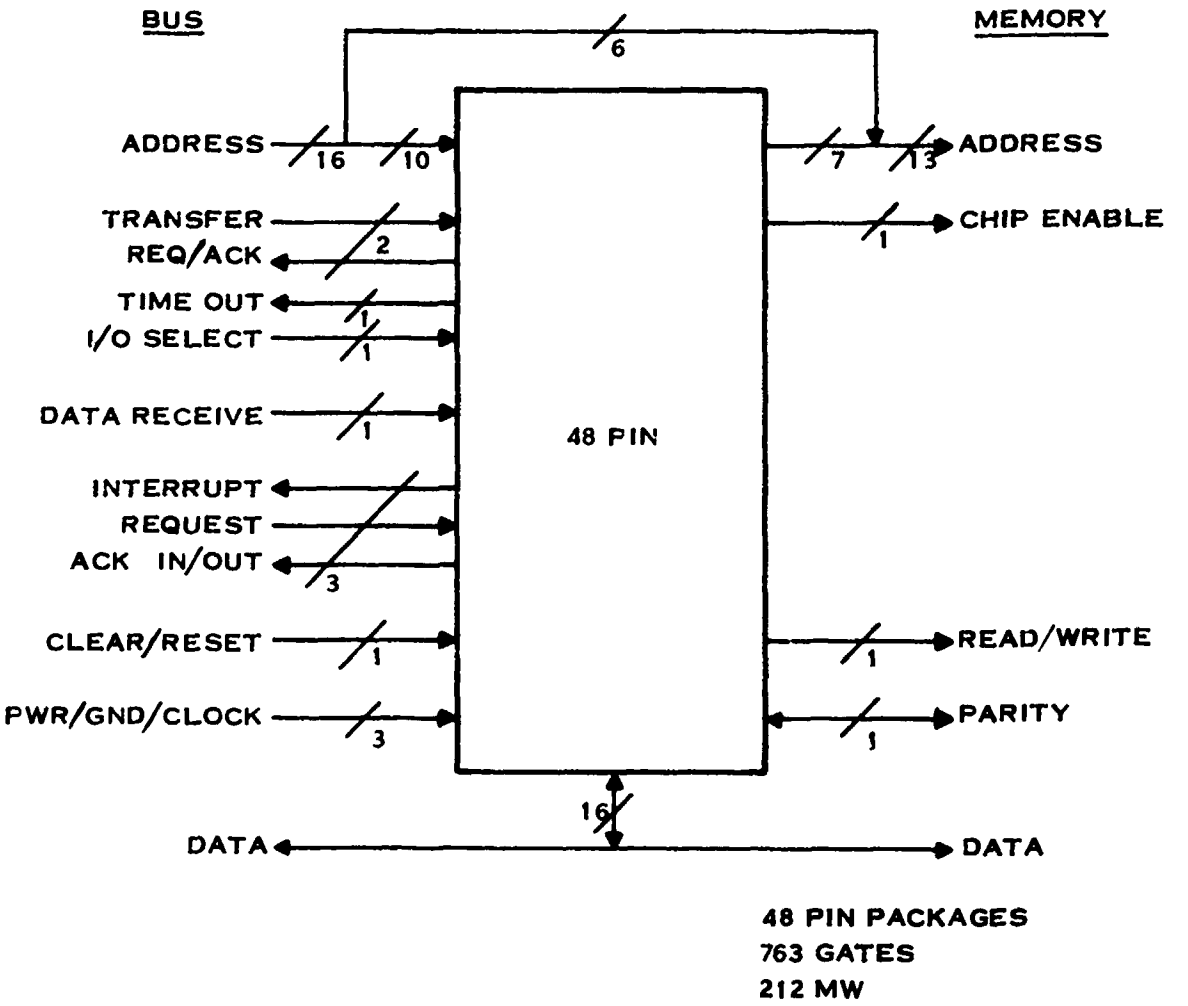



**Figure 84. DP/M Memory Controller**

The I-BUS master clear/reset line is used to command the memory controller to place the memory into the powered-down, or standby, state. This is used in conjunction with an auxiliary power source to allow the data in the memory to be retained during the loss of primary power with minimum power consumption, thus relaxing auxiliary power supply complexity.

The memory control chip generates and checks memory parity and enforces write protection. As discussed in Section IV, the write protection consists of a pair of registers which establish the read/write and the read-only regions. If a parity error or write-protect error occurs, the memory control chip generates an interrupt. When the processor acknowledges the interrupt, the memory controller issues the interrupt trap address over the data lines. The memory control interrupt mask and write-protect bounds registers are accessed by unique I/O device Command Address Words (CAWs). The memory controller is in a 48-pin package and has approximately 763 gates.

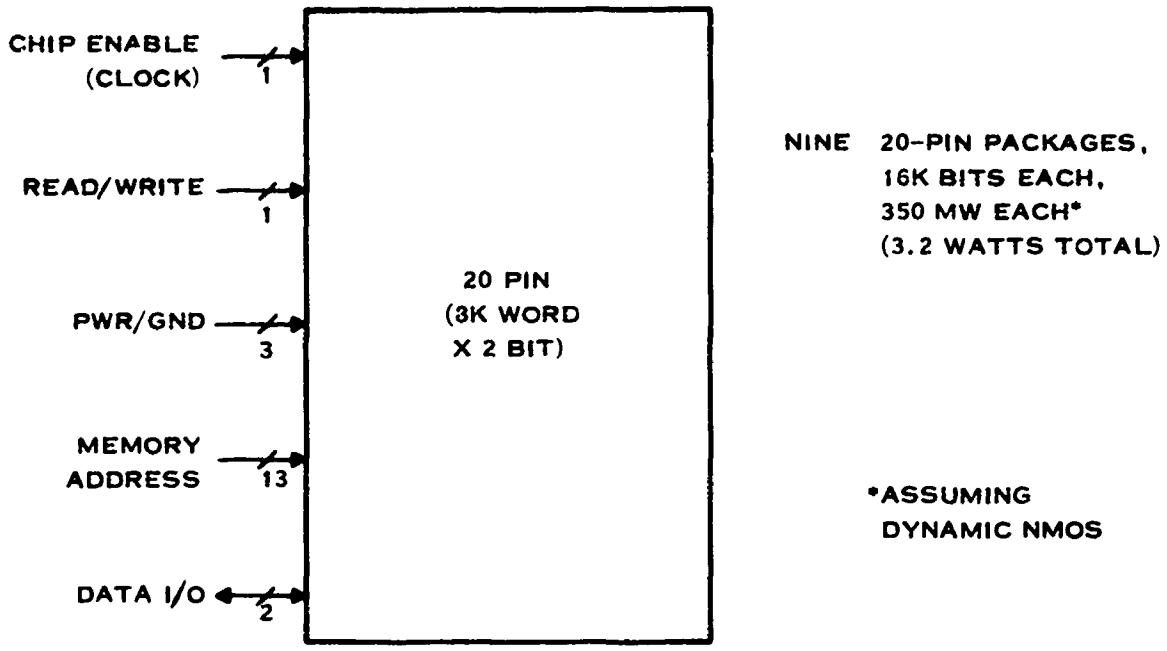



Figure 85. DP/M Memory

### 3. Bus Interface Unit

The advantages afforded by $I^2L$ bipolar LSI technology for the DP/M processor apply to the bus interface unit (BIU) as well. The performance characteristics of $I^2L$ appear more than adequate for the logic operation speeds required for all BIU operations, including the Manchester II encode/decode functions for a 1-Mbps data rate. With the decision thus made to employ $I^2L$ technology in the implementation of the BIU, the final tradeoff consideration is one of physical partitioning and packaging. The two candidate choices are "single-chip" and "multichip" packaging.

Major tradeoffs can be encountered in the area of device complexity and packaging. The approach chosen to partition a functional design into unique physical monolithic devices is primarily dependent upon the following criteria:

Device complexity requirement

- Gate density
- Nonrecurring cost (layout, checkout, debug)
- Recurring cost (yield, producibility, unit test, packaging assembly)

Technology requirements

- Performance
- Environment

Packaging

Commercial compatibility

Recurring cost

Device reliability

Technology requirements can usually be determined at the outset from the device's functional requirements. Major tradeoffs are encountered in the area of device complexity and packaging.

Ultimately, the problem tradeoff becomes what shall be referred to as "ultra large-scale integration," or extremely complex custom LSI single chip implementation, versus partitioning into a set of less complex multiple LSI devices. Evaluation criteria for the DP/M application should be weighed in units of added-value-per-cost requirements for each approach.

Typically, single-chip implementation means large devices for complex implementations such as the envisioned BIU (4,600-gate complexity estimated) with high package pin-out (I/O pins) requirements. In summary, a single-chip BIU would require a large, complex custom LSI device contained within a custom package with device realization optimistically speculated in the late 1970s to early 1980s.

The multi-chip LSI approach possesses merits in all areas where complex, single-chip LSI suffers economic drawbacks. Disadvantages of the multi-chip approach are PE size increase, reduced reliability, and some logistics complexity increase. These disadvantages, however, are judged to present only minor, second-order effects as rationalized below.

Overall PE size may actually be decreased by multiple-chip implementation depending on the larger custom package size in comparison. For example, in present catalog dual in-line integrated circuits, a relatively complex 24-pin package requires more board area than two 16-pin packages with perhaps half the device complexity each; the end result is usually driven by package pin-out requirements. However, these size considerations for the BIU appear to be insignificant because of the small magnitude of the differences involved.

Semiconductor LSI device reliability is roughly proportional to the complexity of the device implementation. Therefore, there does not appear to be a significant reliability difference between the implementation choices because of the number of devices alone. Instead, device reliability becomes a function of physical connections (e.g., solder joints) required to electrically connect the device in the subsystem (PE). Here the single-chip, and hence the single package, usually has a slight advantage because of overall pin-out reductions. However, this advantage is considered minor and is more than offset by the economic advantages afforded by the multi-device approach.

The initial considerations of single-chip versus multi-chip BIU implementation are one facet of the overall physical partitioning approach. It is recommended that the physical partitioning of the BIU produce a BIU device, or set of devices, which is not tied to any one particular bus interface technology/technique. This suggests that all functional elements of the BIU related to unique electrical interface or "language" interface with the TDM buses should be physically segregated from the basic BIU functions for the cost-effective purpose of device (and PE) flexibility and the reduced degree of device obsolescence. Thus, if the "standard" DP/M BIU is

to withstand likely future bus technology/technique modifications (e.g., the replacement of the bus common cable with fiber optics), it should be physically relieved of the bus driver/receiver and bus language translation functional elements. Further implementation considerations should attempt at least a rough-guess prediction of commercially applicable devices for the expected high volume microcomputer/microprocessor market in the DP/M time frame. This area can have a marked impact on device cost if commercial market compatibility is achieved. Finally, consideration should also be given to the implementation time frame desired for the DP/M. Optimally, the functional design should not constrain implementation of the BIU in either LSI approach, and should allow a reasonable, relatively near-term (perhaps interim), implementation realization. Therefore, it is recommended that the BIU design should accommodate and emphasize multichip implementation in accordance with the guidelines discussed previously. Thus, the BIU device partitioning shown in Figure 86 is proposed. The rationale supporting this particular partitioning is presented in the subsequent paragraphs.

The proposed multichip BIU implementation shown in Figure 86 is the result of incorporating all the previously outlined partitioning considerations. The design will require a maximum of four device types for realization of the modular BIU LSI implementation: one of these device types is a non-LSI (SSI complexity) line driver/receiver which may be satisfied with existing catalog parts.

The proposed partitioning first requires segregation of the Global and Local bus interface logic into two physically separate but identical devices (i.e., one common part type) referred to as the Bus Interface Logic Unit (BILU). This is permitted since the functional BIU design requires nearly identical operation by each separate bus interface. There are, however, several minor differences in Global/Local operation which may be handled by a "personality" device-input-signal technique. This concept of the standard BILU device permits the common implementation to perform one of two specific sets of operations based on the binary state of a bus characterization signal connected to one of the device I/O pins. The different operations which must be distinguished within the BILU are:

- Both the global and local interfaces respond to the same type of I/O instruction commands, but each must be addressed (via the I/O instruction Command Address Word) with a different command address; thus the command decode logic must decode different command addresses for global and local operations.
- The Input Message Queue Pointer (IQP) in the global interface must address a different region (queue) in PE memory than that addressed by the local IQP.
- Primary-level interrupt control/interfacing is different for the global/local interfaces (e.g., interrupt trap address generation).

The handling of these differences within a common device by external hardwired control requires very little additional hardware complexity introduced into the common BILU device implementation. This technique does allow the usage of a common part type to implement both global and local interface functions, a characteristic which is important in reducing BIU hardware costs and permitting a near-term implementation realization because of greatly reduced device complexity. In addition, it allows PE configuration with or without both global and local bus facilities, depending on different application requirements. The BILU device will require an estimated complexity of 2,000 gates and may be packaged in a standard 48-pin package. With $I^2L$ implementation, this device will dissipate 606 mW of power. The BILU is by far the most

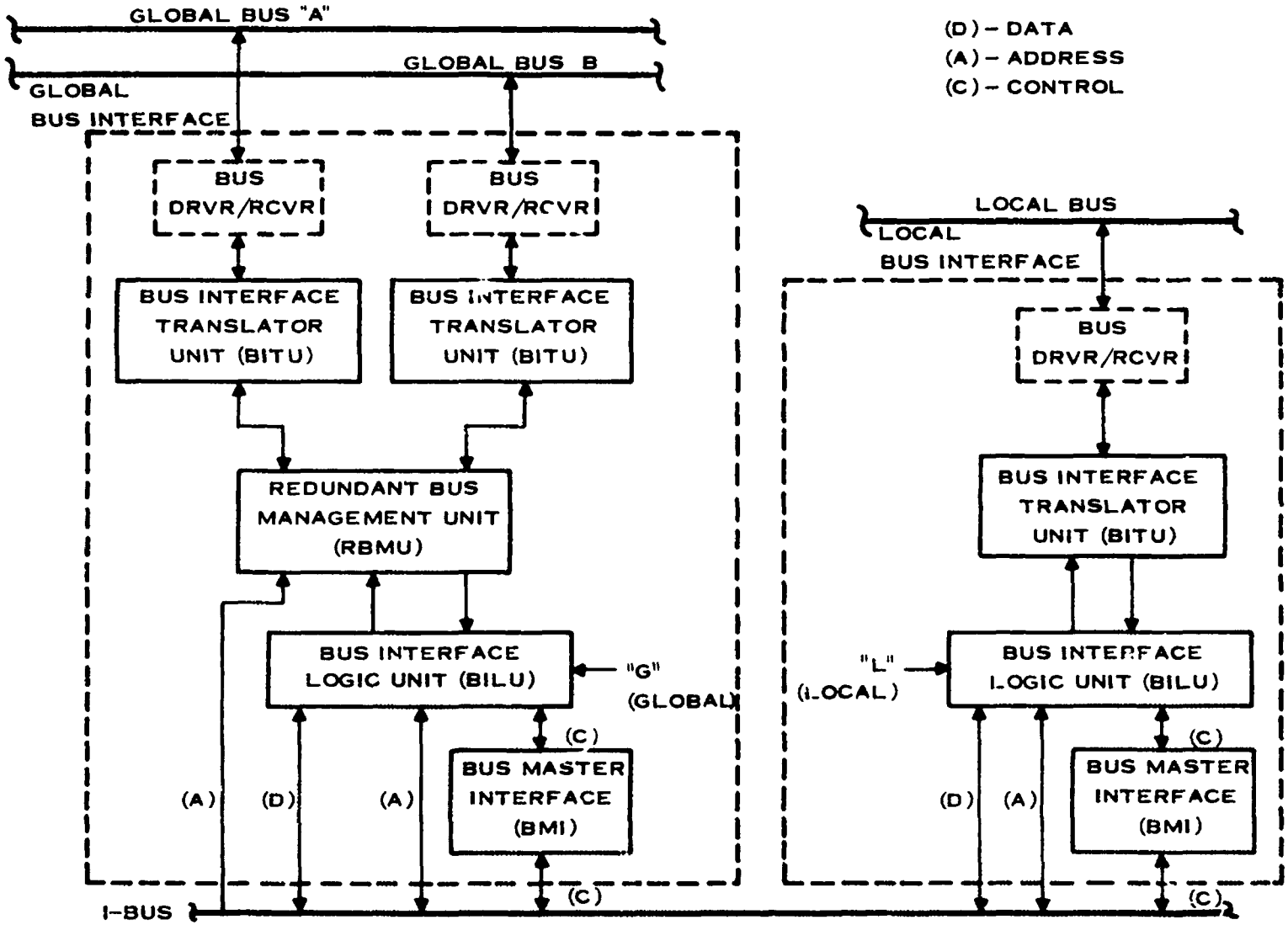



Figure 86. Multi-Device BIU Partitioning

complex BIU device, but its complexity is within the expected near-term realm of $I^2L$ technology. An I/O signal definition of the BILU device is shown in Figure 87.

The suggested partitioning of the BIU removes all functions associated with "language translation" between the Global/Local buses and the PE. The unique device which performs these functions is referred to as the Bus Interface Translation Unit (BITU). This device is effectively inserted in the serial data channel path between a BILU and the bus with which it communicates. It is important that the interface between these two BIU devices should represent a "standardizable" serial interface to accommodate various system bus language requirements with the changing of only one part type, the BITU. The BITU device may be housed in a standard 14-pin package with a complexity of approximately 100 gates. With $I^2L$ implementation, expected power dissipation is only 62 mW. An I/O signal definition of this device is shown in Figure 88.

The remaining BIU device is one which is required to manage the DP/M redundant Global buses. This Redundant Bus Management Unit (RBMU) is functionally placed between the BILU and the redundant bus BITUs. An alternate approach is to place the RBMU between the bus drivers/receivers for both buses and a single BITU. This approach would require one less BITU but would require the inclusion of bus reception translation (e.g., biphase-level to NRZ decoding) logic in the RBMU to allow switchover command decode. This capability inherently exists in the BITU, and, moreover, is responsible for a large majority of the BITU complexity (biphase-level encoding is much simpler to implement than asynchronous biphase decoding). The partitioning approach suggested requires a simpler (less complex) RBMU device at the expense of an added BITU device in the BIU.

The RBMU may be implemented in a single 40-pin device requiring 290 gates. With $I^2L$ implementation, anticipated power dissipation of this device is 41 mW. An I/O signal definition of this device is shown in Figure 89. An alternate and perhaps more desirable partitioning approach allows packaging of the unique RBMU functions in a smaller, standard 20-pin package. An auxiliary device, a standard available quadruple 2-to-1 multiplexer, is required to complete the RBMU function. This alternate partitioning is shown in Figure 90.

The Bus Master Interface (BMI) device shown in Figure 86 incorporates all functional elements related to interfacing the BIU with the PE internal bus (I-BUS) and thus, with the processor and memory units. The BMI device complexity is relatively low; the prime motivation behind its inception is the pin-out relief it offers the BILU, allowing the BILU to be packaged in a commercially compatible package size (48 pins). If further detailed investigation of BILU pin-out requirements allow, the BMI functions may be included in the BILU, assuming the resulting density is attainable in a single device. The BMI functional device implementation is discussed in Subsection VI.C.4 on input/output interface unit (IOIU) device partitioning.

In summary, the BIU partitioning approach presented above stresses near-term LSI implementation compatibility with minimized device cost and allows gradual merging of multiple device functions into single devices as LSI technology progresses, if such measures are deemed cost-effective and desirable. In contrast, implementation of the entire BIU (excluding line drivers/receivers) within a single LSI device, as shown in Figure 91, would require a device complexity of approximately 4,600 gates and 55 I/O signal pinouts. A nonstandard LSI package would have to be used and a rather optimistic furthering of present technology would be required to achieve the required density. Both are very costly, in dollars and risk. The multi-device BIU partitioning approach is presented as the most flexible and cost-effective foreseeable near-term solution to LSI implementation of the BIU function.

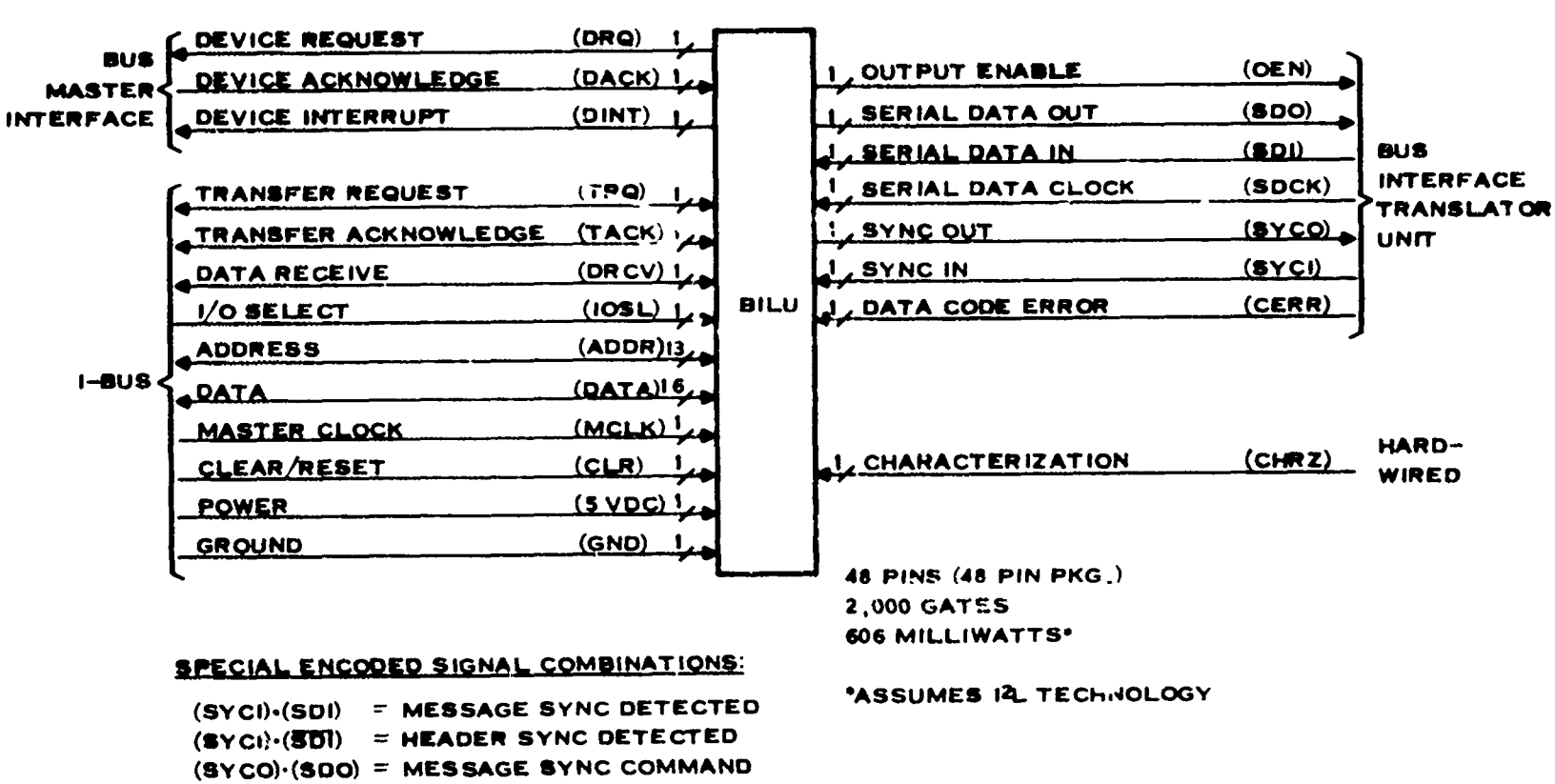



Figure 87. Bus Interface Logic Unit (BITU) Device Attributes

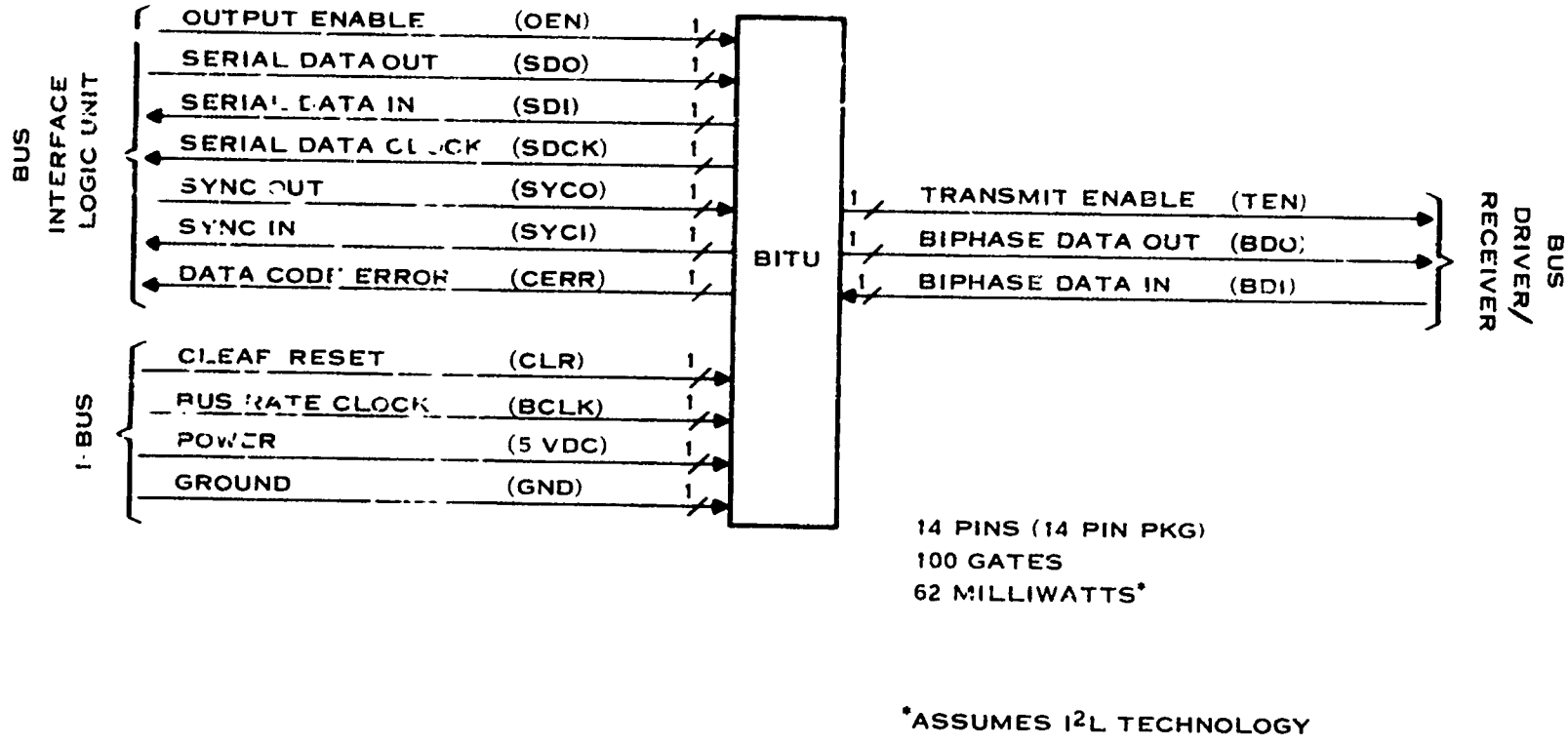



Figure 88. Bus Interface Translation Unit (BITU) Device Attributes

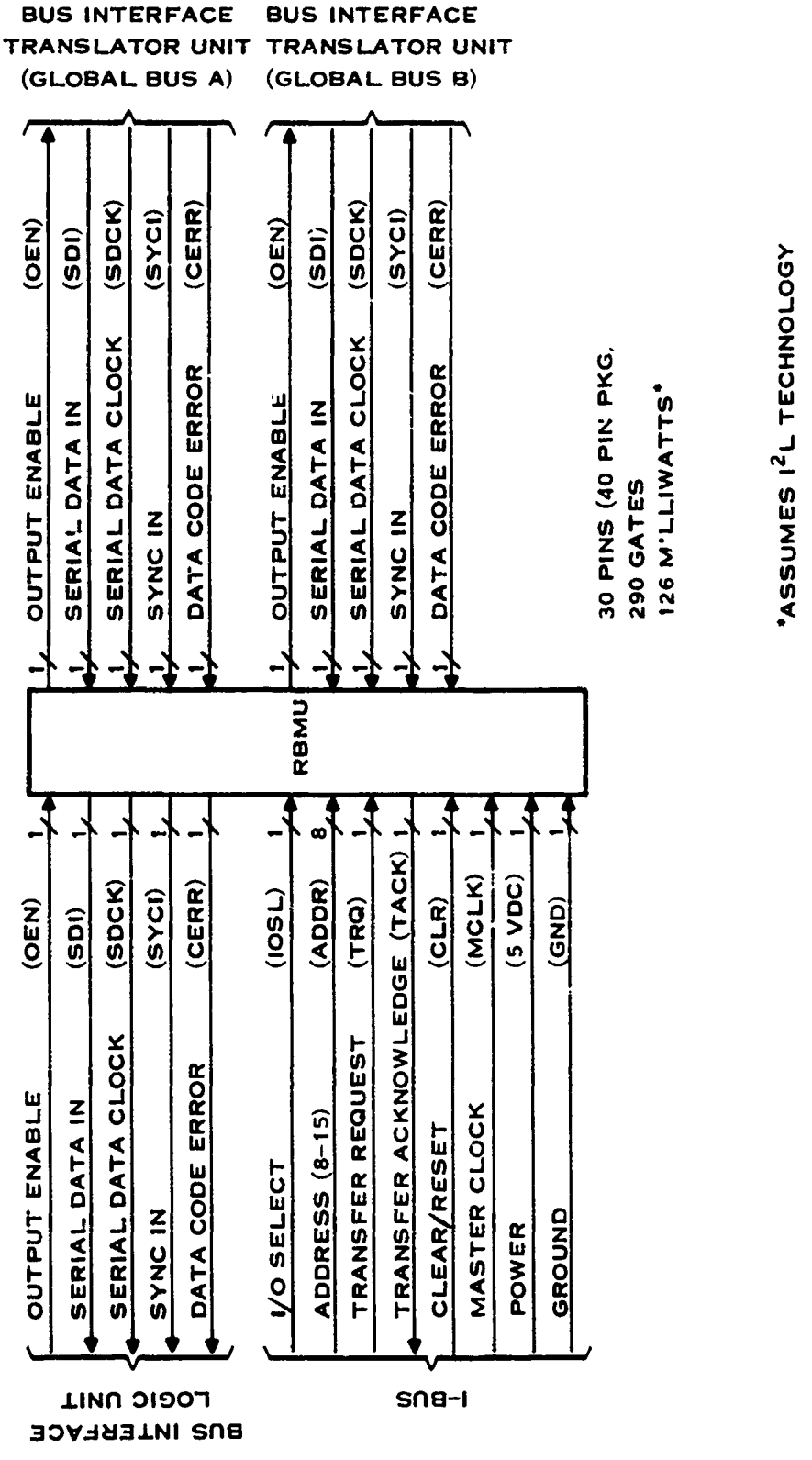



Figure 89. Redundant Bus Management Unit (RBMU) Device Attributes

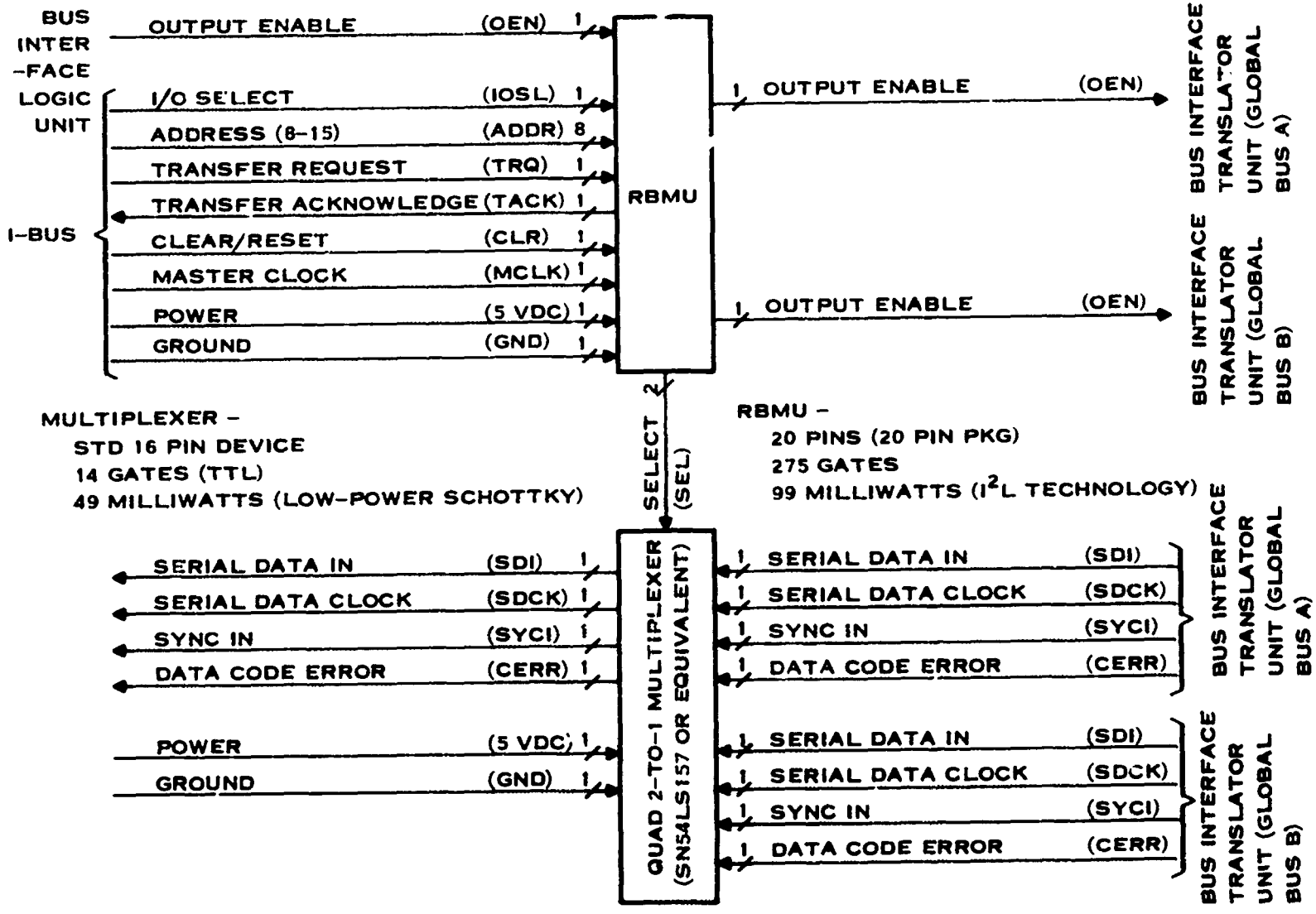



**Figure 90. Redundant Bus Management Unit (RBMU) Device Implementation Alternative**

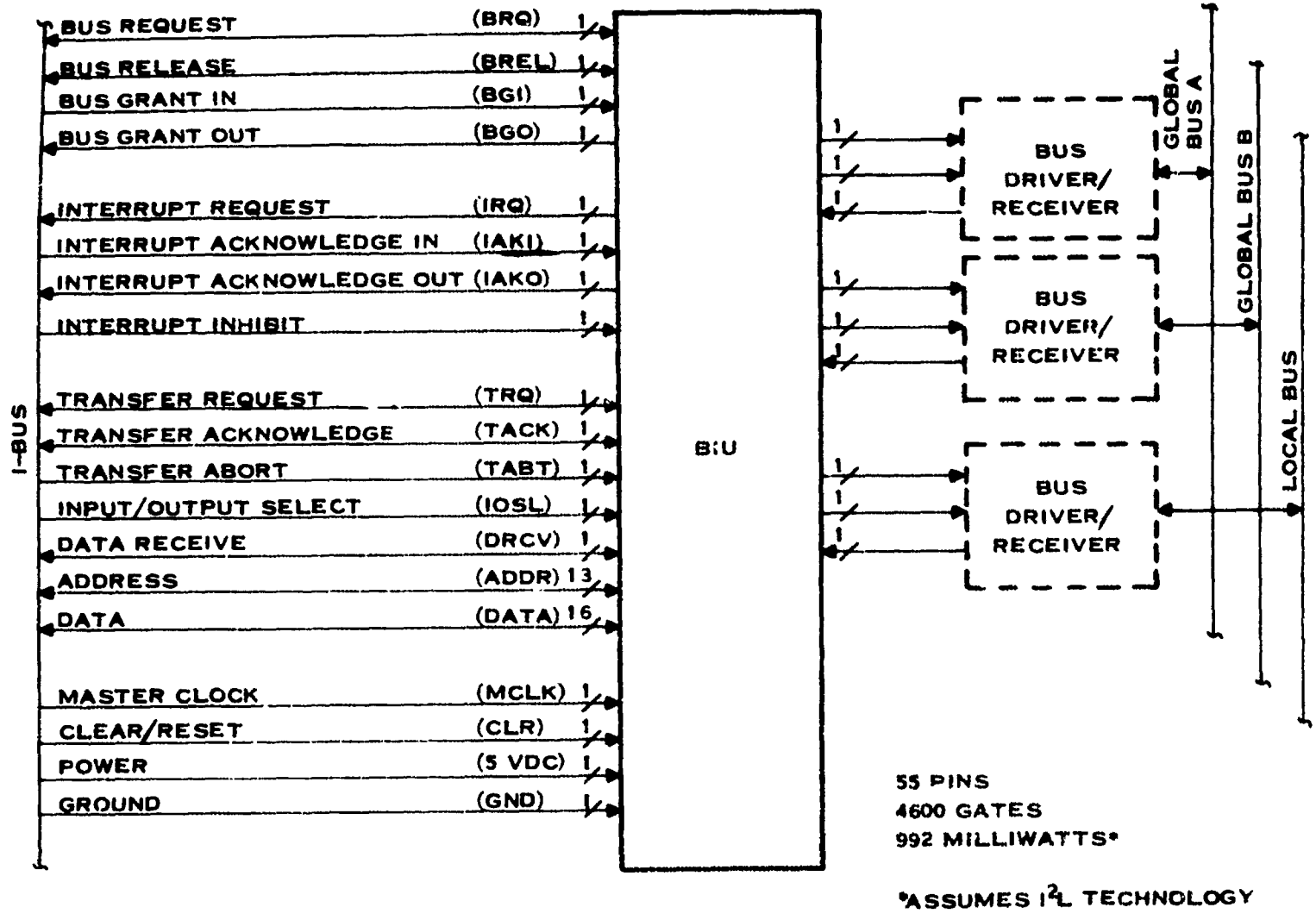



Figure 91. Bus Interface Unit (BIU) Single-Device Attributes

## 4. Input/Output Interface Unit

Initial investigations into physical partitioning and LSI device packaging reveal that the physical implementation of the IOIU will be driven and constrained primarily by pinout (I/O signal) considerations rather than device complexity concerns. As is the case with any parallel I/O interface, more pinouts are always required than are available. This truism holds in the case of the IOIU. A relatively large number of pinouts (71) will be required to implement all IOIU functions with a single LSI device, even though a reasonable device complexity of approximately 1,300 gates is required. If a single-device implementation is desired, then some concession to increased packaging costs must be endured because of the nonstandard pinout requirement (i.e., custom packaging).

To determine the advantages afforded by the use of standard-package LSI parts, a preliminary multichip partitioning was investigated. The initial approach was to relieve the single IOIU device of the pinout burden imposed by the interface with the 16 I/O data lines by buffering these signals with separate Input/Output Data Interface (IODI) devices. The IODI devices provide an output data storage capability and line driver/receiver compatibility between the PE I-BUS and the external I/O data lines. Remaining IOIU functions were retained in a single Input/Output Logic Unit (IOLU) device; however, this partitioning was still unsatisfactory, since remaining IOLU pinout requirements were 55 pins. With a standard device, dual in-line packaging goal of 48 pins maximum, further partitioning techniques were sought. Removal of the ancillary support functions of PE initialization control and PE clock control allows implementation of the remaining, basic IOIU functions in a 48-pin device with a complexity of 950 gates. This level of IOLU device implementation is shown in Figure 92. The segregated support functions would be implemented with standard, available integrated circuit (IC) devices. Using this abbreviated IOLU approach, the physical partitioning of the overall IOIU functional design is shown in Figure 93.

With the realization of the partitioning constraints imposed by pinout requirements alone (with respect to the use of standard IC packaging techniques), further investigation into the advantages of a multichip implementation of the IOLU functions was undertaken. This investigation revealed that the development of an "I/O device family" could yield many benefits with respect to DP/M. The unified bus structure of the PE design both facilitates and encourages the use of modular, building-block I/O construction. Of primary benefit are the following three factors afforded by a "family" structure of common I/O building-block device types:

- Diverse application flexibility. Individual I/O characteristics, which are typically the nongeneral and most difficult to standardize portions of any system design, are easily modified for purposes of system reconfiguration or extended usage adaptation.
- Commercial compatibility. A generalized family of building-block I/O interface devices is directly compatible with commercial multiple application concepts. Since most, if not all, of the IOLU functions defined for DP/M could be considered basic for a majority of applications, a physical partitioning of these functions which allows variable I/O configurations could be significant. If the DP/M PE is to become a cost-effective system building-block device, a primary design goal should be the establishment of parts commonality with a standard commercial production base. Proper structuring and partitioning of the input/output unit functional design could be a key factor in the achievement of this goal. Within the vein of the partitioning philosophy, a functional family partitioning of the IOLU functions was developed. The various device partitionings and the interconnection of these devices into the

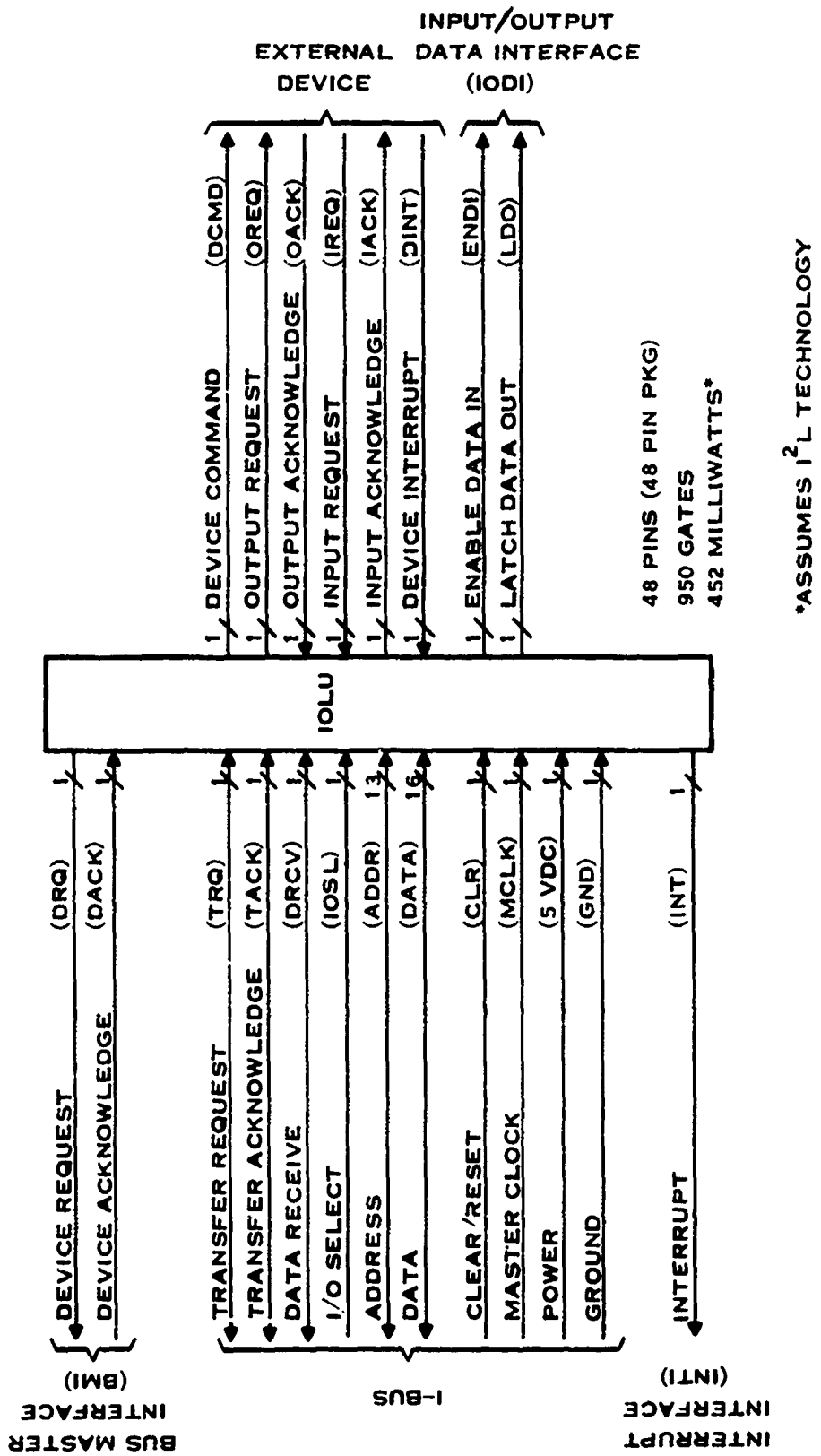



Figure 92. Input/Output Logic Unit (IOLU) Device Attributes

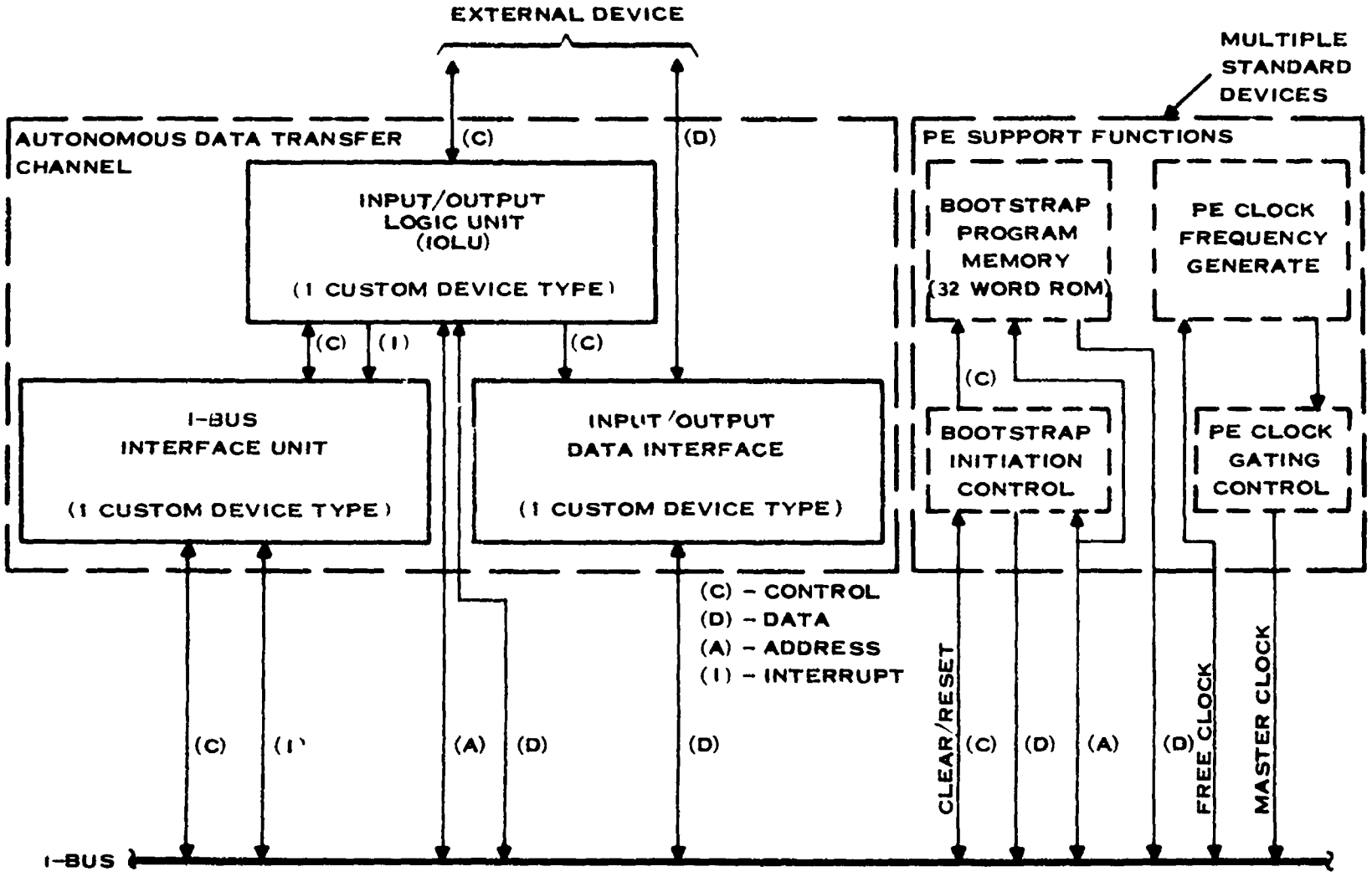



Figure 93. Input/Output Interface Unit Physical Partitioning

DP/M IOLU are shown in Figure 94. The individual functional devices are discussed below. Several functional features are included in the suggested family of devices not representative of the DP/M application per se; however, these features have been included (at insignificant hardware complexity expense) to enhance applicability in a more generalized functional environment.

The Autonomous Transfer Channel (ATC) interface provides an autonomous, block-oriented I/O data channel which possesses the functional characteristics described in Section V. This interface is partitioned into two discrete LSI devices: an Autonomous Transfer Control Logic (ATCL) device and an Autonomous Transfer Control Register (ATCR) device. The ATCL device performs all data channel controller functions; the ATCR provides the necessary register elements required for data channel operation. The two-chip partitioning is dictated by pinout requirements to allow the use of a standard dual in-line packages. Input/output signals and physical characteristics of the ATCL and ATCR devices are shown in Figures 95 and 96.

The IODI device provides data buffering between the external I/O data lines and the PE internal I-BUS. An output data holding register is provided in the IODI to allow asynchronous transfers of data from the PE memory to external devices. Input/output signals and physical characteristics of the IODI are shown in Figure 97.

The BMI device is responsible for providing interface compatibility between a master device and the PE I-BUS. The primary functions of the BMI are I-BUS control protocol compatibility and data transfer timing for either single-word or data-block transfer modes between devices connected to the PE I-BUS. For system design convenience, the BMI also provides transfer time-out determination with hardware-selectable timing. The BMI device serves the same function as the IBIU device mentioned in the previous BIU partitioning investigation. Input/output signals and physical characteristics of the BMI device are shown in Figure 98.

The Bus Slave Interface (BSI) device provides a straightforward and convenient means of interfacing slave I/O devices to the PE via the I-BUS. The BSI can support either responsive or nonresponsive communication with these devices. The BSI provides address decode and read/write operation determination associated with slave device data transfer operations. The BSI allows selectable (hardwired) address definition for the slave device which it interfaces and selectable timing of slave device data transfer control signals. These selectable characteristics permit the BSI device to accommodate virtually any type of passive I/O device including most conventional semiconductor memories. An I/O signal level description of the BSI device is given in Figure 99.

Functionally, the BSI is used to achieve programmed I/O operations between the processor and I/O devices where one word of data is transferred to/from the device per I/O instruction execution. In DP/M applications where this mode of I/O data transfer is desirable, programmed I/O may be achieved by direct connection with the associated I/O device (and BSI device) via the I-BUS, as illustrated in Figure 94.

The Interrupt Interface (INTI) device provides a flexible interrupt stimulus interface to the PE via the I-BUS interrupt-associated signal lines. Basic INTI functions include:

- Interrupt request and acknowledge logic/timing
- Interrupt priority resolution

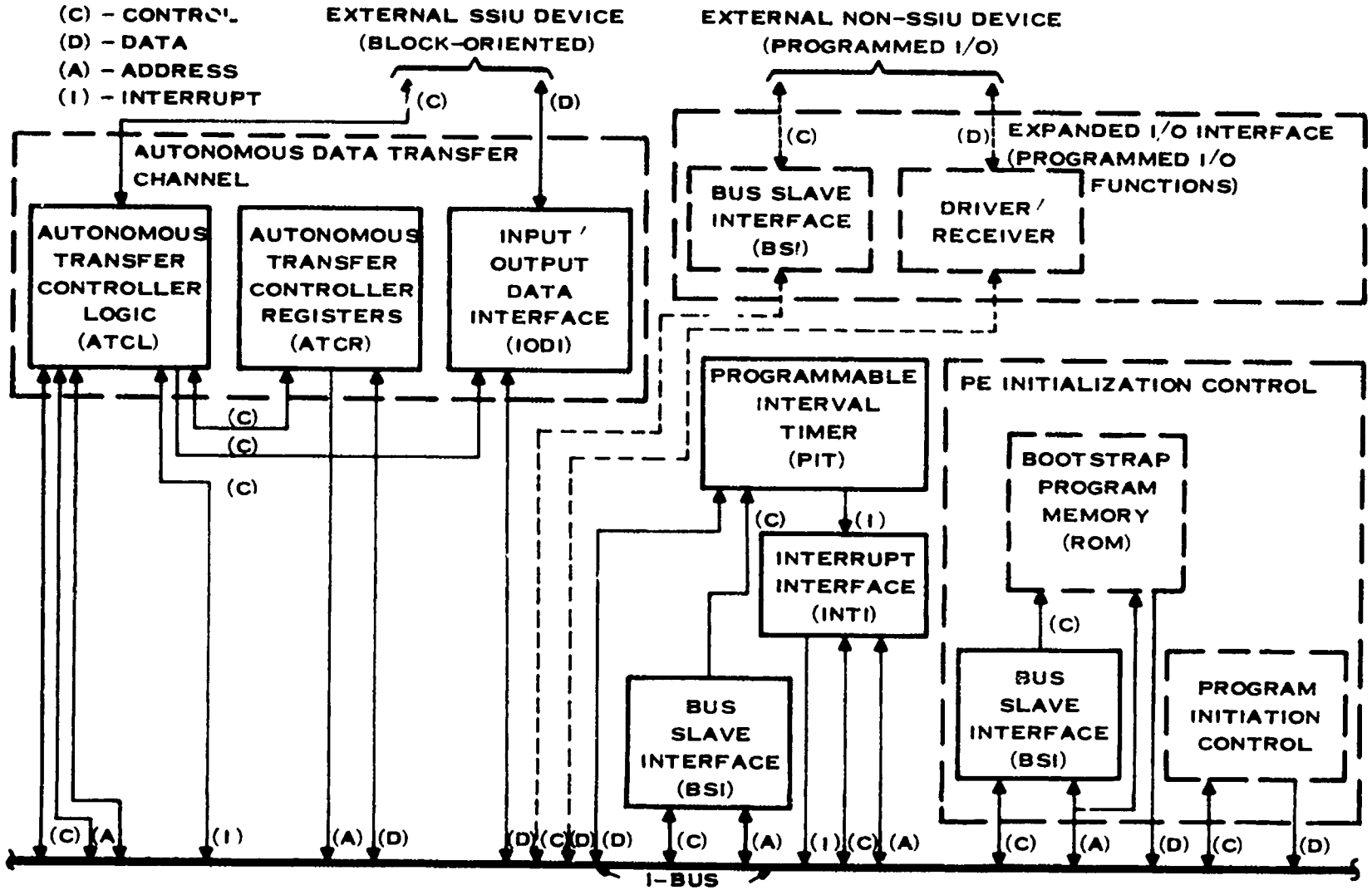



Figure 94. Input/Output Interface Unit Modular Device Family Partitioning

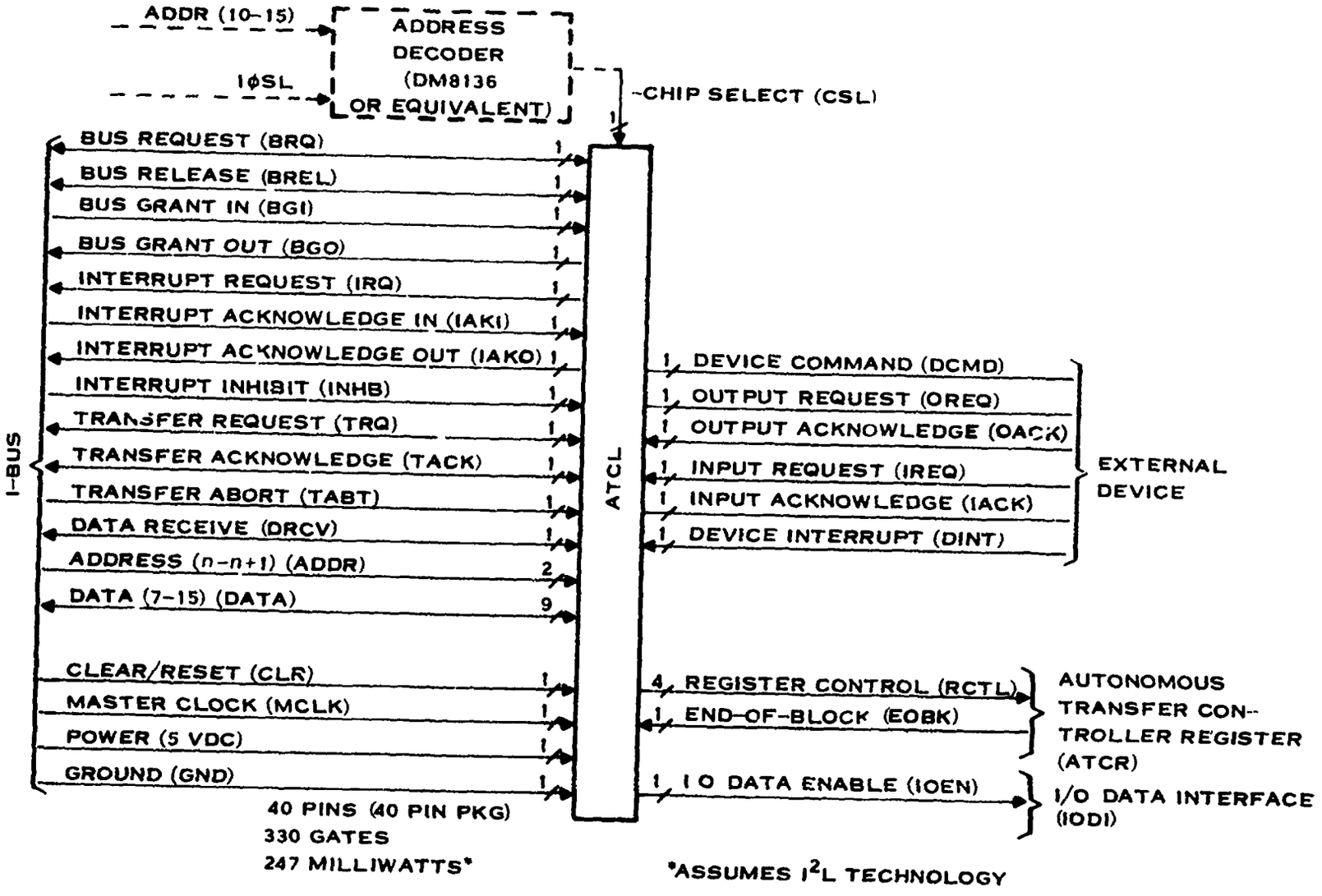
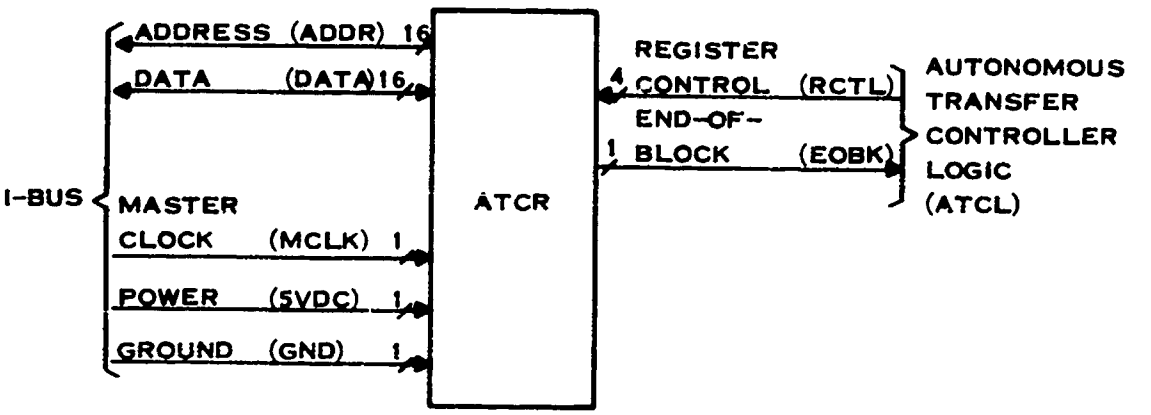



**Figure 95. *Autonomous Transfer Controller Logic (ATCL) Device Attributes***




FUNCTIONAL REGISTERS
(EACH 16-BITS IN LENGTH)

BUFFER ADDRESS REGISTER (BAR)
BUFFER LENGTH REGISTER (BLR)
LINK ADDRESS HOLDING REGISTER (LAHR)
INTERRUPT TRAP ADDRESS REGISTER (ITAR)

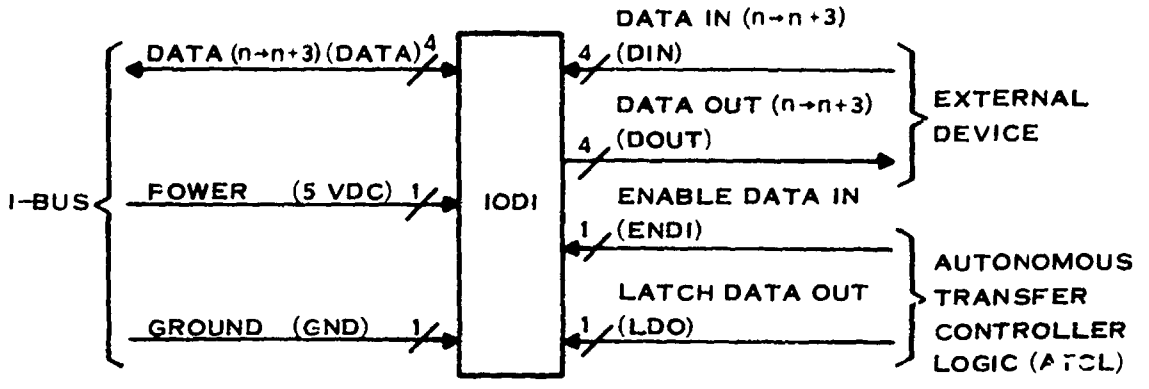
40 PINS (40 PIN PKG)
755 GATES
352 MILLIWATTS*

*ASSUMES $I^2L$ TECHNOLOGY

REGISTER CONTROL ENCODED FUNCTIONS:

| | RC0 | RC1 | RC2 | RC3 | ACTION |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | DO NOTHING |
| PROGRAM-CONTROLLED READ | 0 | 0 | 0 | 1 | BAR → DATA |
| | 0 | 0 | 1 | 0 | BLR → DATA |
| | 0 | 0 | 1 | 1 | LAHR → DATA |
| PROGRAM-CONTROLLED LOADS | 0 | 1 | 0 | 0 | DATA → ITAR |
| | 0 | 1 | 0 | 1 | DATA → BAR |
| | 0 | 1 | 1 | 0 | BAR → ADDR |
| | 0 | 1 | 1 | 1 | DATA → BAR, BAR → ADDR |
| MEMORY READS | 1 | 0 | 0 | 0 | DATA → BLR, BAR → ADDR, BAR+1 → BAR |
| | 1 | 0 | 0 | 1 | DATA (08-10) → BLR, BAR → ADDR, BAR+1 → BAR |
| | 1 | 0 | 1 | 0 | DATA → LAHR, BAR → ADDR, BAR+1 → BAR |
| MEMORY WRITES | 1 | 0 | 1 | 1 | DATA → LAHR, BAR → ADDR, LAHR → BAR |
| | 1 | 1 | 0 | 0 | BLR-1 → BLR, BAR → ADDR, BAR+1 → BAR |
| | 1 | 1 | 0 | 1 | DATA → BLR, BAR+1 → BAR |
| | 1 | 1 | 1 | 0 | LAHR → BAR |
| | 1 | 1 | 1 | 1 | UNDEFINED |

Figure 96. Autonomous Transfer Controller Registers (ATCR) Device Attributes




16 PINS (16 PIN PKG.)
34 GATES
100 MILLIWATTS*

*ASSUMES LOW POWER SCHOTTKY TECHNOLOGY

FUNCTIONAL LOGIC DESCRIPTION

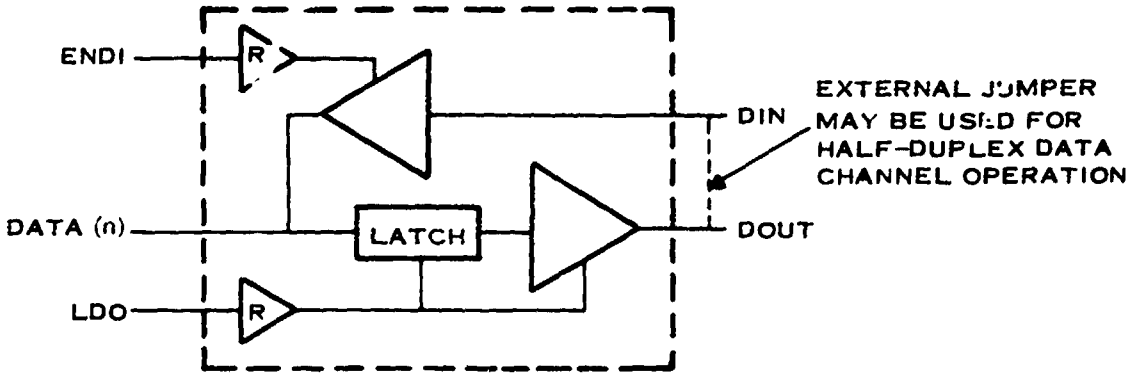



**Figure 97. Input/Output Data Interface (IODI) Device Attributes**

- Programmable interrupt mask (1-bit)
- Programmable interrupt stimulus reset control

Input/output signals and physical characteristics of the INTI device are shown in Figure 100. The Programmable Interval Timer (PIT) function described in Section V can be readily implemented with a single low-complexity LSI device. To permit PIT compatibility with a wide variety of applications, the suggested single LSI device design possesses some additional features above and beyond those required for the DP'M avionics application investigated. These additional features are:

- Timer resolution is programmable to permit a selectable count period of

$$\frac{1}{f_{clk}} \text{ to } \frac{1}{f_{clk}} \times 2^{-15}$$

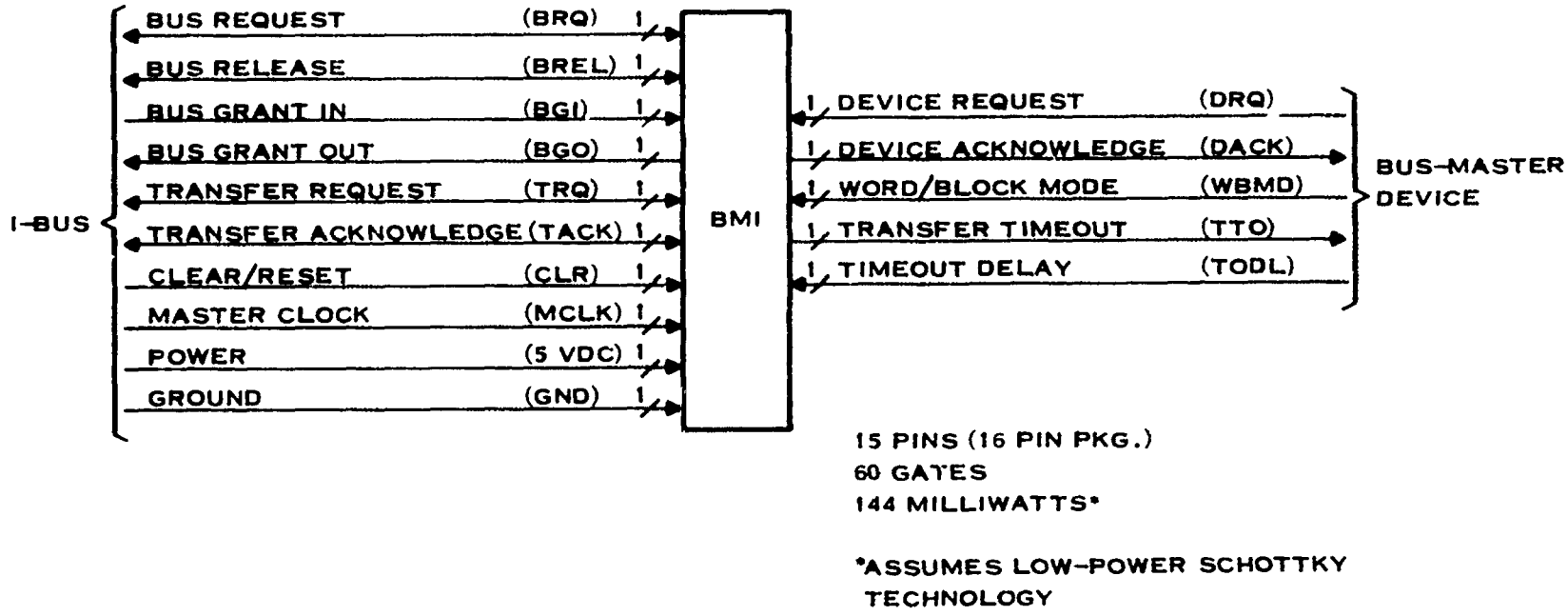
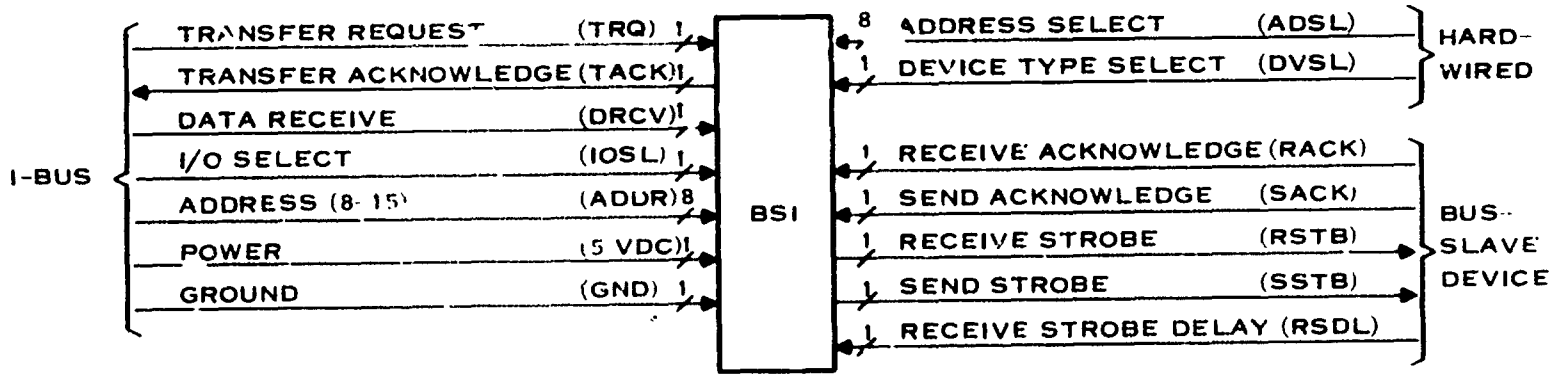



Figure 98. Bus Master Interface (BMI) Device Attributes




28 PINS (28 PIN PKG)
32 GATES
76 MILL "ATTS"

*ASSUMES LOW-POWER SCHOTTKY TECHNOLOGY

**Figure 99. Bus Slave Interface (BSI) Device Attributes**

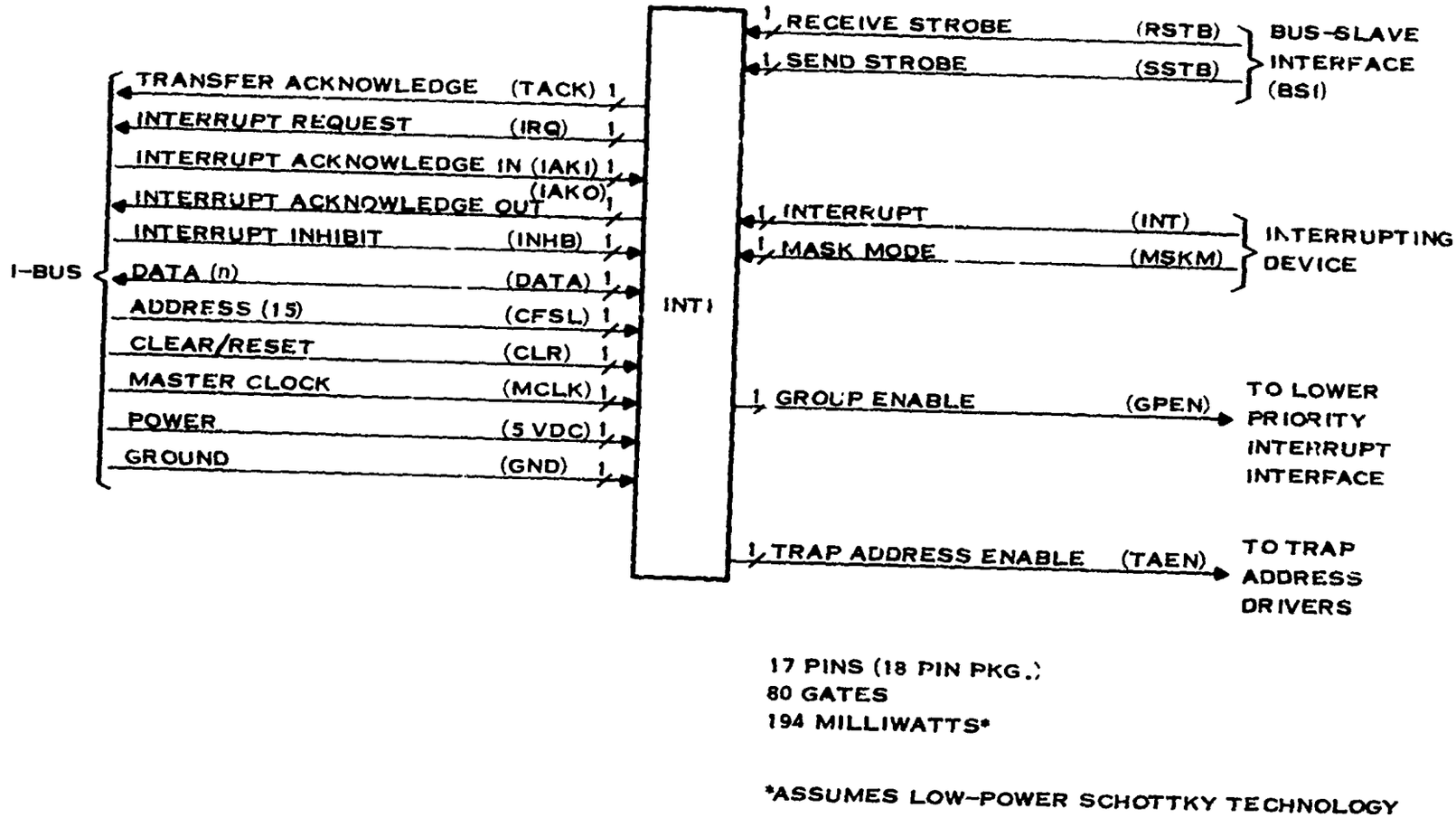



Figure 100. Interrupt Interface (INTI) Device Attributes

where $f_{clk}$ is the input clock frequency.

- Additional control logic allows hardwired selection of device function as a programmable interval timer or real-time-clock ("time-tick" source) and allows cascading of PIT devices to achieve larger time interval capabilities.

These additional features will require increased device complexity, but the magnitude of the increase is not considered significant with respect to added device cost. An I/O signal definition of the suggested PIT device is given in Figure 101.

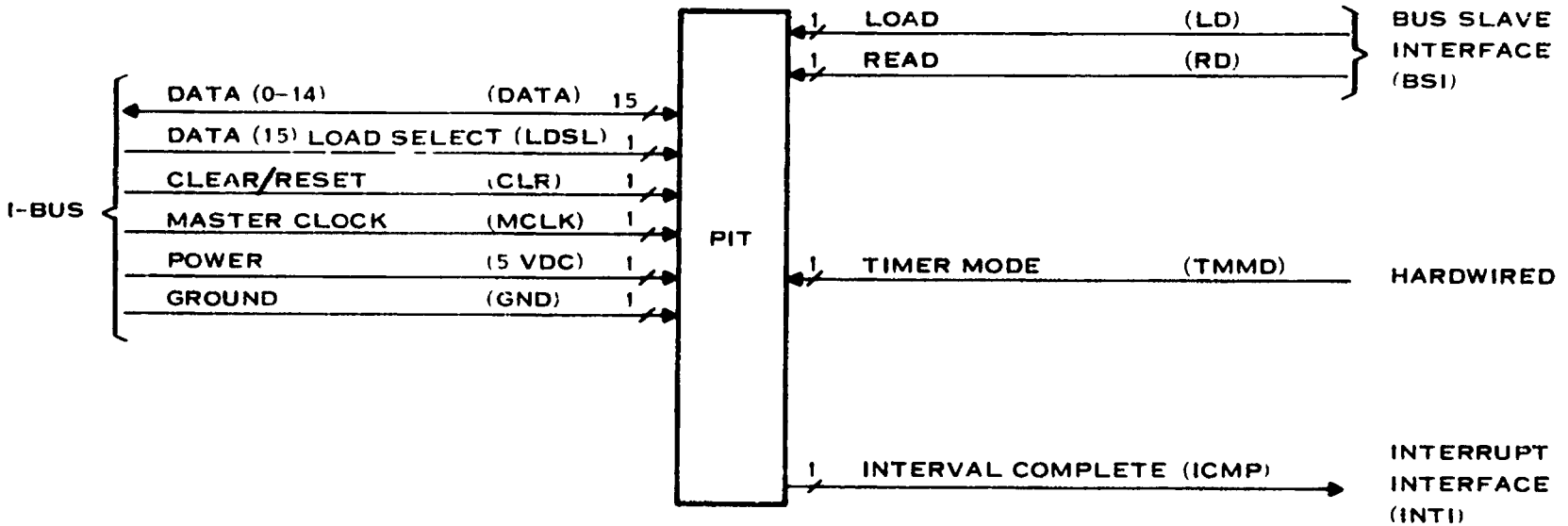
The above "device family" partitioning results in a set of low-risk complexity LSI and low complexity (MSI) devices. The MSI caliber devices (i.e., the IODB, BMI, BSI, and INTI devices) allow implementation in a standard, conventional, non-LSI technology, such as currently proliferating low-power Schottky bipolar technology, without an overbearing increase in power dissipation while enhancing overall device applicability because of improved device performance capabilities and potentially decreasing device costs associated with development and production.

Remaining IOIU functions not provided in the above family of devices are PE initialization control and PE clock control. Since both of these PE support functions may be straightforwardly implemented with a few standard (catalog) integrated circuit part types, the fabrication of special, unique LSI devices to be used for their implementation is not considered cost effective and, thus, is not recommended.

The PE initialization function requires a 32-word by 16-bit ROM for a bootstrap program memory plus the control logic required to sequence memory address steering to the bootstrap program address. From a physical viewpoint, the bootstrap ROM (probably two IC devices) may be considered and partitioned as part of the PE memory. The bootstrap program steering functions may be implemented with a few available small-scale integration (SSI) IC devices.

The clock control function requires the necessary counter circuitry to divide down an externally-sourced "free" clock frequency to the basic clock frequency usable by the PE. This PE clock must be gated in response to a maintenance-function-related clock control signal to provide RUN/STOP modes of operation. To prevent the PE design (functional and physical) from becoming tied to a given technology, it is recommended that all PE clock frequency derivation elements be physically partitioned outside the basic PE. By placing these devices external to the PE proper, improvements in PE performance through technology advancement or alteration may be accommodated without requiring redesign (relayout) of any PE LSI devices. An analogous consideration applies to the clock gating function as well, because of possible unique maintenance function clock control requirements. It is recommended that all clock control functions be segregated from IOIU LSI device implementation, regardless of whether the single-device or multi-device IOIU partitioning approach is accepted.

The briefly discussed family of I/O devices and resultant IOIU physical partitioning are deemed to possess the functional and physical properties representative of a truly comprehensive, universally applicable I/O building-block structure. These devices allow basic interfacing with many I/O device types and offer an expandable "pay-as-you-go" system design using the DP/M PE. It bears repeating, however that this particular device partitioning approach is not required entirely for the DP/M application per se, but it affords the cost-effective virtues of system device flexibility and likely commercial parts base compatibility, factors to be weighed with overall DP/M PE design goals. The use of commercially available part types and the promise of




24 PINS (24 PIN PKG)
490 GATES
211 MILLIWATTS*

*ASSUMES $I^2L$ TECHNOLOGY

Figure 101. Programmable Interval Timer (PIT) Device Attributes

near-term implementation with overall reduced LSI parts costs (simpler devices) justifiably guide the IOIU design toward the multi-device partitioning described herein.

## E. COST PROJECTIONS

It is admittedly difficult to make accurate cost projections as far into the future as 1980. The cost figures presented herein are for production units only and assume a quantity order of 5,000 units. They do not include any design, development and/or qualification costs nor any factors for inflation. The basic factors that have been used for estimating are that a high-volume standard-production, dynamic NMOS memory cell will be used; it will cost about 1 cent/bit; and the processor will cost about 1 cent/gate. Both of these estimates are based on commercial grade parts (0° to 70°C, plastic package). Because the memory controller, the I/O interface, and the bus interface are not as universally applicable, they are expected to have a lower volume production. It will be assumed that they will cost twice as much per gate (i.e., 2 cents/gate). For comparative purposes, projections have been worked up for three different versions of the DP/M PE (Table 23). All cases assume an external power source.

### 1. Commercial Microcomputer

The commercial microcomputer version uses commercial plastic parts (0° to +70°C, no device acceptance test burn-in), a 4K word memory without parity or write protect, no bus interface unit, and a two-sided printed wiring board (PWB) with a one-piece (edge) connector.

**TABLE 23. PROJECTED 1980 DP/M PE COSTS**

| | Commercial Microcomputer | Commercial Grade DP/M PE | Military Grade DP/M PE |
|---|---|---|---|
| Temperature range | 0° to +70°C | 0° to +70°C | −55°C to +125°C |
| Burn-in | No | No | Yes |
| Packaging (DIP) | Plastic | Plastic | Ceramic |
| Memory: | | | |
| Size (1024s) words | 4 | 8 | 8 |
| Parity | No | Yes | Yes |
| Write protect | No | Yes | Yes |
| Connector (PWB) | one-piece[a] | one-piece[a] | two-piece |
| Cost | | | |
| Processor | $ 40 | $ 40 | $200 |
| Memory | 64 | 144 | 720 |
| Memory controller | 5[b] | 15 | 75 |
| I/O interface | 30 | 30 | 150 |
| Bus interface unit | | 100 | 500 |
| PWB, connector and assembly | 20 | 20 | 100 |
| Total | $159 | $349 | $1,745 |

[a] PWB edge connector
[b] Refresh only

This yields a "bare bones" 16-bit microcomputer for under $200, and these figures are somewhat lower than projections made by others. For example, Herman Schmid speaks of 16-bit CPUs for $50 and 16-bit microprocessors[1] (electronics only) for $200 both in quantity and in 1980. As was anticipated, the memory tends to dominate, amounting to almost half the cost of the microcomputer. It is noteworthy that this cost corresponds to between a 10-to-1 and 20-to-1 reduction in cost over contemporary minicomputers which have the same level of processing capability.

### 2. Commercial Grade DP/M PE

The next step in the cost projection is to start adding features to the "bare bones" microcomputer to upgrade it to a DP/M PE. This involves doubling the memory (already the most expensive item), adding parity and write protect to it, and finally adding the BIU which is more complex than the processor (CPU). These additions more than double the cost of the microprocessor and bring its cost for a commercial grade DP/M up to between $300 and $400. These figures are well in line with the figures projected by Honeywell in their DP/M study[2] especially when considering that the memory has been doubled in size to allow better use of the processing power of the PE. Again these costs are between a 10-to-1 and a 20-to-1 reduction over what one would currently pay for a minicomputer with these capabilities.

### 3. Military Grade DP/M PE

The biggest single factors in the cost of the DP/M PE are the military environmental and reliability requirements. All devices must be premium parts which can operate over the full military temperature range of −55° to +125°C. These are the requirements for components to be used in equipment which must meet a MIL-E-5400, Class 2, environment. This specification requires equipment to operate continuously at −54° to +71°C with intermittent (30-minute) operation at +95°C. This latter condition would only allow a 30°C thermal gradient between the external LRU environment and the semiconductor devices inside the equipment. In addition, the reliability requirements dictate many in-process inspections, hermetically sealed ceramic packages rather than plastic, and usually an extensive burn-in and testing cycle. These additional requirements normally increase the component cost by 5 to 1. Other factors that increase the PE cost are the additional PWB fabrication inspections and controls, as well as the requirement for two-piece military-qualified connectors. These factors send the cost of the DP/M PE into the $1,500 to $2,000 range. While this figure may seem high, it is still 0.05 to 0.1 the cost of most current military-qualified computers. This 5-to-1 premium for military-qualified semiconductor components and equipments has to do with the lower volume of military procurements coupled with the cost of the added processing steps and the requirement for premium parts.

### 4. Cost Conclusion

The DP/M PE will allow between a 10-to-1 and a 20-to-1 decrease in the cost of computers for the military avionics application. It will not allow the 100-to-1 decrease in cost that some have projected. The reason for this 5-to-1 difference is that the military requirements for operating temperature and reliability are much more severe than the commercial requirements. If the commercial requirements became more like the military, or vice versa, this difference would disappear. An example of a commercial application that has a military-like requirement is the automobile. Similar high-volume, rugged applications like this could bring the military-grade cost down to nearly the same as the commercial-grade cost.

[1] Herman Schmid, "Monolithic Processors," *Computer Design*, October 1974.

[2] Honeywell, Inc., "All-Semiconductor Distributed Aerospace Processor/Memory Study," Volume II, Technical Report AFAL-TR-73-226, August 1973.

# SECTION VII

# FUNCTIONAL SIMULATION SYSTEM

## A. SECTION OVERVIEW

This section is concerned with the Functional Simulation Task (SOW 4.1). In particular, this section describes tasks to be performed, design approach taken, design results, and key features of the resulting simulation system.

## B. INTRODUCTION

Within the context of the DP/M system concept, the objective was to design, develop, and use the necessary simulation and analysis tools adequate for evaluating the major design considerations of a DP/M network. The Simulation System is divided into two separate simulators: the System Network Simulator and the Processing Element Simulator. Each simulator will be discussed in detail in the following paragraphs, however, a general description of each will be included at this time. The System Network Simulator is a high-level traffic simulator whose main function is to study the effects of different topological network organizations. The Processing Element Simulator is a more detailed simulator concerned with the internal performance of a single processing element. The PE Simulator actually simulates functions performed as each instruction is executed by the Processor.

The nature of the DP/M architecture requires certain capabilities of the eventual Functional Simulation System. The design simulator approach was strongly influenced by the characteristics of the DP/M system to be simulated. In particular, the DP/M system consisted of a network of autonomous processing elements connected by two levels of time-division-multiplexed buses. Control of the buses was distributed as well as some portion of the executive control. The Processing Element itself was similarly constructed with processor, memory, I/O, and bus interfaces all communicating over an internal bus. In addition, there were requirements for variable levels of simulation detail. Some events needed to be modeled at the register and/or clock level, while other events were sufficiently modeled at a higher functional level. For example, the Bus Interface Unit was modeled at the detailed hardware register level in the Processing Element Simulator but the bus was modeled at the functional level in the System Network Simulator. To satisfy these goals, the Functional Simulation System was structured as a discrete event-oriented simulation system built around a nucleus of common model independent utility routines. The model independent routines are the tools by which the actual simulator is constructed. The model independent routines by themselves are not a simulator, but are used to build the different simulators. The simulation control is implicit within the basic simulator and explicit control of the events is not required. The use of a common simulation language allows a uniform method of development, implementation and modification. The two simulators are functionally compatible, allowing for interaction and interchange of data.

Key features of the simulation tools developed include a common uniform structure written in ANS FORTRAN that also permitted ease of modification and future growth. The System Network Simulator can be considered to be a continuation of capability. Although the System Network and Processing Element Simulators are logically separated for the purpose of discussion, they are actually the same type of simulator with different events modeled to a particular level of detail. Each has the capability to model variable time quantum events and

variable levels of detail. The most valuable feature by far though, is the extensibility of the simulation system from the initial hardware design on through the software development and system integration. For example, the models for the avionic software within the System Network Simulator may be replaced by higher and higher fidelity models until the model represents the actual assembly language that then may be simulated by the Processing Element Simulator. In the course of these evolutionary simulations, the system under development goes through a metamorphosis in the sense that one starts with the high-level simulation of the entire system. Gradually, high-level models of system components are one-by-one replaced with their more detailed counterparts. Once functional simulation has been completed, implementation and simulation of the actual software model begins. Finally, the entire system with all of the actual application software is being simulated. This method of system development provides flexibility with a minimum investment in actual hardware and maximum possibility of ultimate system success.

## C. SIMULATION APPROACH

### 1. Defintion of Terms

In order to remove any ambiguity concerning simulation terminology, the following terms will be utilized in describing the DP/M Simulation System. A *system* is a collection of functionally and logically related entities, each characterized by attributes which, in turn, may be related among themselves. An *entity* denotes an object of interest in a system. An *attribute* denotes a property of an entity. An *event* signifies a change in a state of an entity. For example, an aircraft and its pilot can be considered to constitute a system; the entity "aircraft" has the attributes of type, cost, etc.; the entity "pilot" may be described in terms of his age, qualification, etc. Relationships between entities may be classified into *static relationships* and *dynamic relationships*. Furthermore, dynamic relationships can be expressed as *deterministic functions* of time or as *stochastic functions* of time, or as a mixture of both.

### 2. Simulation Control Structure

The DP/M Functional Simulation System is a discrete event-oriented simulation system built around a nucleus of common model independent utility routines. Unlike a continuous system where transitions from one state to the next are a continuous function of time, within a discrete system, transitions from one state to another occur at discrete time points. In discrete systems, distinguishable state transitions are called events. Thus, an event can occur only at specific times, and there are no changes between events; furthermore, an event is an idealized phenomenon of zero duration. Event-oriented simulation systems emphasize a detailed description of the steps that occur when an individual event takes place.

Simulation control structures within a discrete event-oriented simulator are represented by Figure 1C2. The Simulation Control Algorithm (SCA) controls simulation time, maintains the Future Event List, and provides any ancilliary system routines. The heart of the simulator is the Future Event List. The Future Event List (FEL) is a chronologically ordered list that contains event notices. Associated with each event notice is an activity routine that simulates the actions of the particular event to be modeled. The SCA removes the first notice from the FEL, advances simulation time to the time associated with the event, determines which activity routine is associated with that event and passes control to it. Simulation time may be advanced to the time associated with the next FEL entry since the FEL is chronologically ordered, thus there can be no event before the first element on the FEL. Thus, time is determined by the sequence of

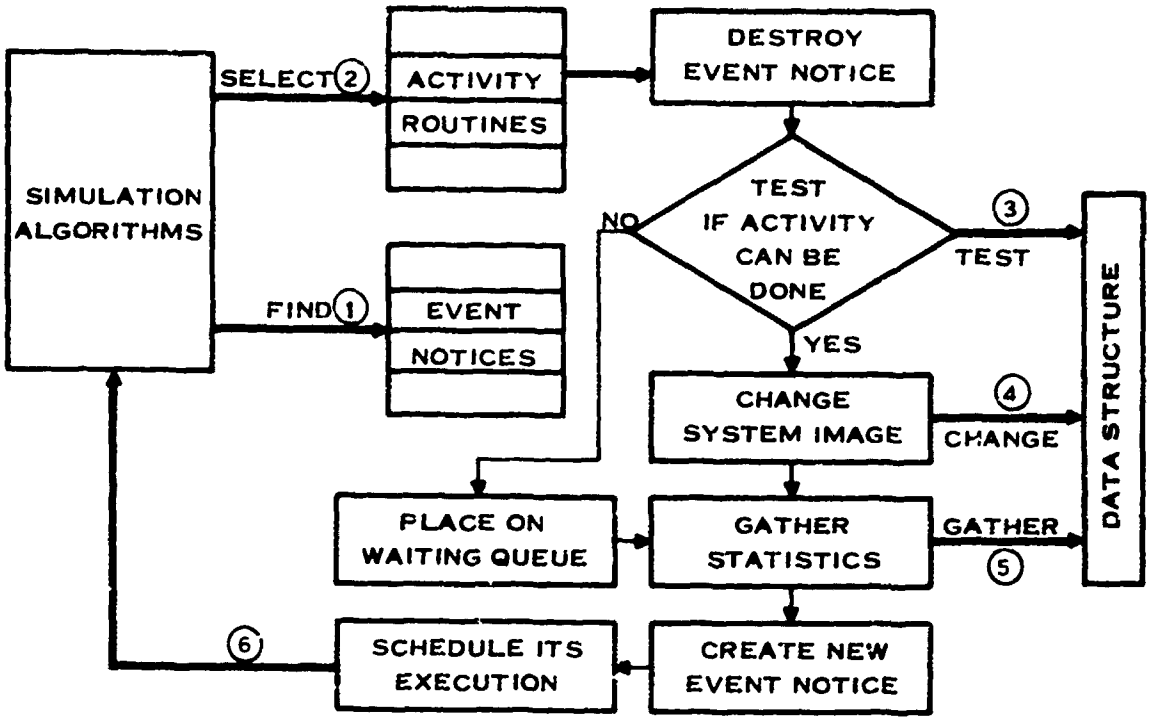



**Figure 102. Simulation Control Structure**

events and not by some fixed interval. This characteristic is sometimes referred to as a variable time increment or next event approach.

Each activity routine performs essentially the same task. First, it destroys its event notice, after extracting any pertinent information, by returning it to dynamic memory. Next, the activity must test, via its data structures, if all precedence relationships have been satisfied. If not, the activity is placed on a waiting queue until such time that the constraints are satisfied. If all constraints are satisfied, the activity may be executed, thus changing the state of the system. Necessary statistics are then gathered. Finally, a new event notice is generated, a time for the event to be executed in the future is computed, and the event notice is returned to the SCA to be placed on the FEL. The SCA will then remove the next event from the FEL and proceed in the same manner as before.

### 3. Basic Simulator

In order to efficiently implement the simulation approach previously described, certain useful characteristics were identified and package of routines that provides a powerful tool for the development of discrete event-oriented simulation systems was developed. This Basic Simulator is characterized by model independence, FORTRAN orientation, dynamic memory management, list processing facilities, clock management, random number generators and error diagnosis and reporting.

*a. Model Independence*

Many of the functions performed within the simulation approach are completely independent of the actual system to be simulated. In order to make the Basic Simulator applicable to a wide range of potential simulation problems, as much of the package as possible was generalized to provide the tools to develop a discrete event-oriented simulation system without precise knowledge of the system to be simulated.

*b. FORTRAN Orientation*

The Basic Simulator approach offers several important features: (1) the user need not learn a new (specialized) programming language, i.e., it uses FORTRAN as a base language; (2) all standard facilities of the host operating system, on which the Basic Simulator is installed, are readily available to the user in a familiar way; (3) it is possible to develop a simulation system that may be transported to other computer systems that have compatible FORTRAN compilers and operating system capabilities.

*c. Dynamic Memory Management*

Dynamic memory management of storage required by the simulator provides the capability to manipulate during program execution a vector of common memory so that blocks from that vector can be temporarily allocated to the user's program and then deallocated when no longer needed. This memory management method makes it possible to reuse the same storage many times for different purposes throughout a simulation run and, thus, not only minimize total program storage requirements, but also increase the size of the simulation problem that can be handled.

At any instant the memory vector consists of two types of blocks: (1) those that are allocated and currently in use; (2) those that are deallocated and, thus, are available for use. All deallocated blocks are chained together as a linked list, which is called the Free List. When a program requests a block of memory the Free List is searched to find a deallocated block of sufficient size. The search/allocation strategy works as follows: (1) during the search, physically adjacent blocks in the Free List are collapsed into a single large block; (2) the requested block is allocated from the first sufficiently large free block encountered in this search. If no such block can be found, further execution of the simulation program is terminated.

The Dynamic Memory consists of the following major routines:

MEMALX Allocate Memory This routine is used to allocate a block of memory from the Free List.

MEMFRX Deallocate (Free) Memory This routine is used to free an allocated block of memory and to return it to the Free List.

MEMZOX Store Zeroes in Memory This routine is used to store zeroes in all user available words of a specified memory block.

*d. List Processing Facilities*

In order to allow efficient representation of the data structures within the simulator, as well as to provide flexible methods of modification to those data structures, some type of list

processing facility is required. In addition, list structures, in particular queues, allow for gathering of useful statistics concerning system variables. For example, to measure the loading of a communication bus, data are placed on a bus queue for the length of time that would be required to deliver that data to some receiver, and removed by the receiver sometime later. Queue statistics are automatically gathered that provide such information as minimum/maximum size, mean time on queue, standard deviations, etc. Thus, list structures provide efficient means of measuring system parameters.

List processing facilities developed for the Basic Simulator are used for defining, searching, adding, deleting elements and maintaining standard list statistics for doubly-linked lists.

The List Processing Facilities consist of the following major routines.

LTDEFX Define a List
This routine creates a standard list head and then stores into it information which defines and initializes the new list.

LTADDX Add an Element to a List
This routine adds a new element to the correct logical position in the list identified by the list-head pointer.

LTFNDX Find a Specific Element in a List
This routine searches a list in the direction of decreasing rank of its elements to find an element which contains a specific value stored in a specified word.

LTNXTX Return a Pointer to the Next Element in a List
Given a pointer to an element in a list, this routine returns the pointer to the element which is logically next to the first pointer.

LTRSPX Remove a Specific Element from a List
This routine removes a specific element from the specified list.

### *e Clock Management*

The Clock Management portion of the Basic Simulator makes it possible to simulate concurrent processes by representing each process as a sequence of events. The flow of control during simulation is managed by means of a time-ordered list, called the Future Event List (FEL), which contains the event notices for events scheduled to occur at some future time. The Clock Management consists of the following major routines.

SCHDLX Schedule an Event
This routine places an event notice on the Future Event List.

ENTRYX Calculate the Entry Address to a Subprogram
This function, coded in assembly language, is used to compute the run-time entry point address of a FORTRAN subroutine.

CANCLX Cancel a Future Event Notice
This routine cancels a future event notice.

### *f. Random-Number Generators*

Many phenomena within the DPM system are stochastic in nature. In order to be able to accurately model these random processes, random-number generation capability was provided

The random-number generators developed provide facilities for producing sequences of random numbers from standard statistical distributions most commonly used in simulation, as well as for generating random numbers from continuous and discrete distributions defined by the user

g *Error Diagnosis and Reporting*

The error diagnosis and reporting capabilities developed provide the user with tools for debugging simulation programs In particular, three basic types of tools are provided (1) those for writing diagnostic messages and for optionally halting further simulation execution (2) those for checking the validity of operational parameters and subroutine arguments relative to Dynamic Memory and list processing operations (3) those for dumping (e g printing) the contents of the data aggregates or structures typically used in simulation, such as various portions of Dynamic Memory, lists or vectors Those checks performed in number two are user controllable, and may be suppressed if not required to increase simulation efficiency

## D. SYSTEM NETWORK SIMULATOR

### 1. System Network Simulator Models

The DP M System Network Simulator is basically a traffic simulator used to study the topological considerations of the DP M network As such, it is a high-level simulator The corresponding models that form the simulator are concerned with forming a structure with emphasis on variability of those design parameters at the DP M network level In particular, the System Network Simulator was used to analyze and evaluate (1) processing element characteristics capabilities, (2) number of resources in the system, (3) local and global bus configuration, (4) inter-PI communication technique, (5) PI Bus protocol communication technique, (6) executive control technique To fulfill the goals of the DP M System Network Simulator the following models were developed (1) Bus Control Model (2) Avionic Function Program Models, (3) Executive Models

The avionic function program models describe the execution of an avionic program for each PI in terms of the time to execute the modeled avionic program, memory words required, and messages generated for the local and global buses by that program Avionic program execution times for a PI are determined from the execution time estimates for individual tasks generated by the system requirements effort The execution times are described in units of basic processor operations per second The message traffic generated for the system communication buses is derived by the system requirements effort

The executive model simulates in detail the scheduling algorithm of the executive The executive scheduling algorithm determines which of the tasks are to be executing at any one instant in time The bus control model simulates the message sequencing protocol activities of the Local and Global buses The performance of the buses is modeled in terms of the bus allocation algorithm Each of these models are discussed in detail in subsequent paragraphs

### 2. System Network Simulator List Structures

In order for the System Network Simulator to be efficient and effective it must allow variable DP M configurations to be evaluated without a major modification to the simulator itself For example, one series of evaluation might be concerned with studying the effect of

different partitioning of tasks within a fixed DPM configuration. While another might be concerned with modifying the DPM configuration itself.

To allow for ease of change of configuration or partitioning of the DPM system hardware and software, a flexible list structure system was designed. These list structures allow two levels of definition. One at the logical level and another at the physical level. The designed list structures define (1) terminals, (2) IPC interconnectivity, and (3) logical precedence relationships between tasks.

## 3 Terminal List Structure

The terminal list structure in Figure 103 provides the mechanism to describe each of the terminals in the DPM system. A terminal is any physical device within the DPM System. A

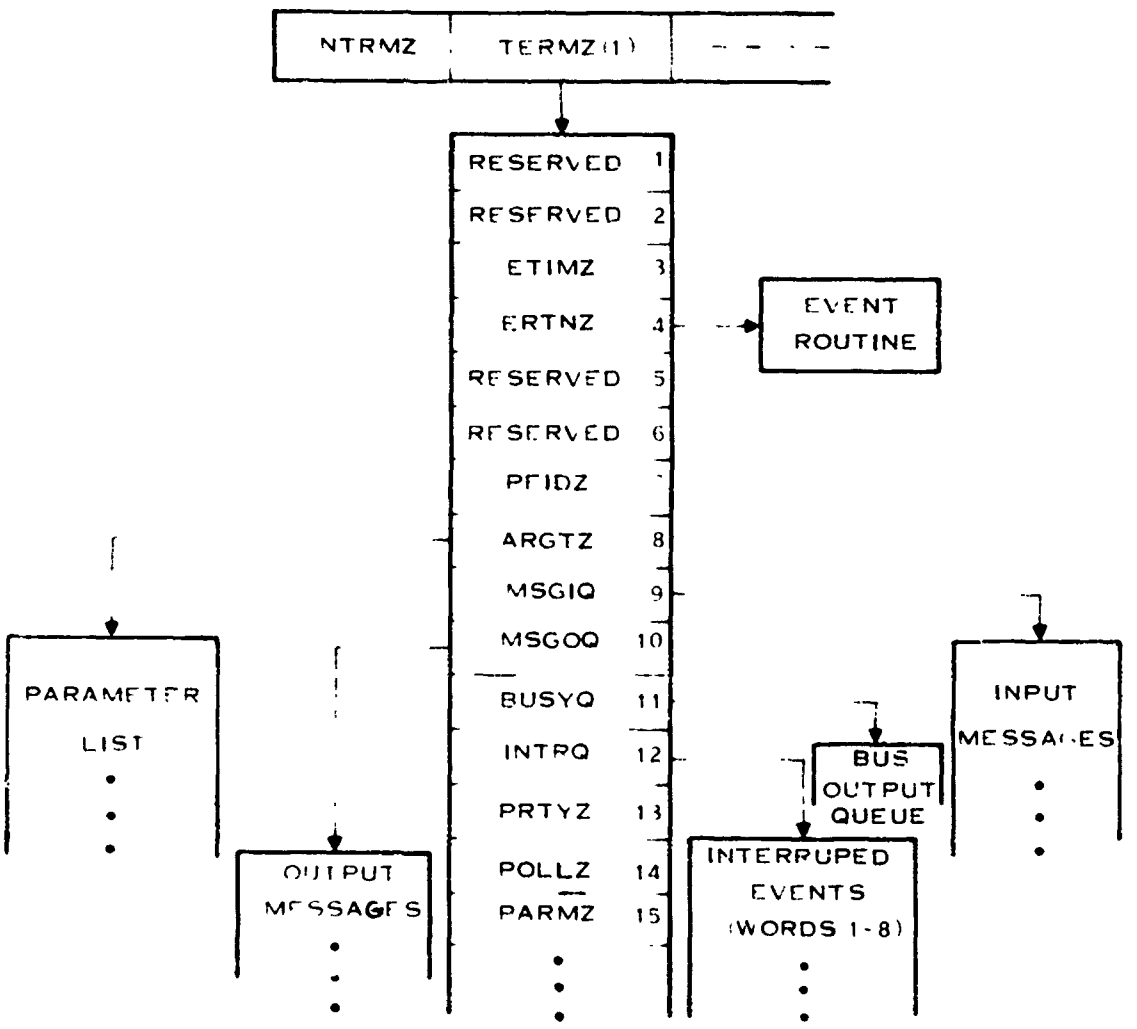



Figure 103 Terminal List Structure

terminal includes such things as processors, sensors, bus controllers, interrupt generators, etc. Each terminal device has a list structure as defined in Figure 103. The first seven words correspond to the format of a standard event notice. PEIDZ contains the processor identity number. ARGTZ contains a pointer to a list of parameters that are uniquely associated with the current event routine. MSGIQ contains a pointer to a queue that contains any input messages required by that terminal. MSGOQ contains a pointer to the queue that contains any output messages required by that terminal. BUSYQ contains a pointer to a queue that contains any output messages from the terminal that are currently on the bus. INTRQ contains a pointer to a queue that contains any events that have been interrupted and have not completed. PRTYZ contains the priority of the current event routine. POLLZ contains the polling period if the terminal device has a polling loop. PARMZ is the first word of a list of parameters that are uniquely associated with the terminal device.

## 4. PE Interconnectivity

The PE interconnectivity list structure in Figure 104 provides the mechanism to describe and vary the physical connectivity of the Local and Global buses without modification to the simulator. For each affinity group, a Local bus connectivity list as well as an entry in the PE-affinity group association list is generated. Likewise, a global bus connectivity list is generated describing the global PE interconnectivity. Current Position of Control (CPC) pointers are initialized to the first PE within each list. At the completion of each message transmission over a bus, the CPC pointer is advanced. All messages are checked to verify that the destination PE is actually connected to the transmitting PE. The PE-Affinity Group association list is utilized to locate the correct Affinity Group for a transmitting PE.

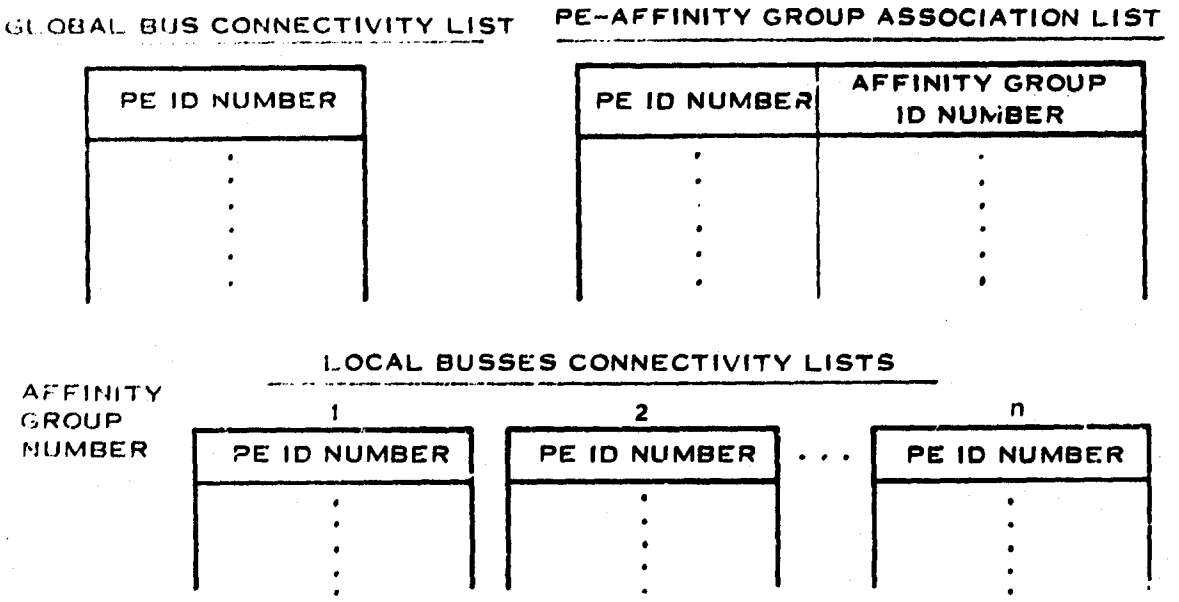



Figure 104. PE Interconnectivity List Structure

## 5. Logical Precedence Relationships Between Tasks

In order to provide flexibility within the System Network Simulator so that various physical organizations may be evaluated, a necessary requirement is that logical precedence relationships between tasks be representable independently of physical considerations. This separation of logical and physical relationships can be accomplished by use of the following procedure: (1) describe system logical precedence relationships as directed graphs, (2) describe tasks in terms of pertinent parameters, (3) represent message (data) transmittal to processors by input and output queues, (4) pass all output message requests to a virtual message distributor that is cognizant of the actual method in which data is to be transmitted.

## 6. DP/M Test Mission Characteristics

Using the previously discussed guidelines, a hypothetical test mission was formed which closely resembled an air-to-ground Close Air Support (CAS) quick-reaction mission against short-life targets such as tanks or trucks. The set of functions was selected to allow some of the more demanding digital processing loads for DP/M testing and analysis, and are not intended to represent any one given or complete avionic configuration for a particular mission. For example, the inclusion of an inertial strapdown navigation system in a CAS avionics suite is unlikely; however, an inertial strapdown system was included because it poses a more demanding computational load on DP/M than does an ordinary gimbaled inertial navigation system.

Eight classes of mission functions were defined for DP/M analysis. These eight mission functions include:

(1) Navigation

(2) Landing

(3) Air Data

(4) Flight Control

(5) Fire Control

(6) Vehicle Defense

(7) Stores Management

(8) System Management.

Associated with each of these general functions are definite modes of operation called subfunctions. For example, within the navigation function there are the subfunctions of Loran, Inertial Strapdown, as well as various alternate and backup modes of navigation operation. This classification of functions and subfunctions closely resembles the way in which the pilot or mission manager views his avionic system. A third level of definition is required for the system architecture and digital processor designer and this is the task level of definition. A task is a logical unit of the subfunction, and is usually associated with a well defined activity or process within the subfunction, such as the receiver interface task within Loran. For the most part, discussion concerning DP/M mission processing will center on subfunctions and their associated tasks. A summary of the DP/M mission functions, subfunctions, and tasks that were defined is given in Table 24.

## TABLE 24. DP/M SYSTEM PROCESSING TASKS

| Function | Subfunction | Task | ID |
|---|---|---|---|
| Navigation | Loran | Receiver Interface | P 11 |
| | | Bearing Angle Expected Time Difference | P 12 |
| | | Time Difference to Lat Long | P 13 |
| | | Delay Time Correction Velocity Information | P 14 |
| | | Universal Transverse Mercator Output Display | P 15 |
| | Inertial Strapdown | Gyro Compensation | P-21 |
| | | Acceleration Compensation | P 22 |
| | | Attitude Integration | P-23 |
| | | Coordinate Transformation and Velocity Accumulation | P-24 |
| | | Navigation Calculations | P 25 |
| | | Reference Rotation | P-26 |
| | | Dead Reckoning Calculation | P 27 |
| | Kalman Filter | Covariance Matrix Update | P 31 |
| | | State Vector Update | P-32, 33 38 39 |
| | Air Mass | Air Mass | |
| | Wind Estimate | Wind Estimate | |
| | Steering | Steering | |
| Air Data | Air Data | Mach No Free Air Temperature Calculation | P-41 |
| | | True Air Speed Calculation | P 42 |
| | | Altitude Calculation | P 43 |
| Flight Control (AFCS SAS) | | Pitch Stabilization Augmentation | P 51 |
| | | Pitch Autopilot | P 52 |
| | | Roll Stabilization Augmentation | P 53 |
| | | Roll Autopilot | P 54 |
| | | Yaw Stabilization Augmentation | P 55 |
| Fire Control | NARBS | Impact Point Calculation | P-64 |
| | | Partial Derivatives | P 65 |
| | | Impact Point Prediction | P 66 |
| | | Lateral Steering | P 67 |

## TABLE 24. DP/M SYSTEM PROCESSING TASKS (Continued)

| Function | Subfunction | Task | I.D. |
|---|---|---|---|
| Fire Control (Cont) | | HUD Preprocessing | P-68 |
| | | SARBS Filter | P-69 |
| | FLIR Control | Ground Stabilization | |
| | | Track Handle Control | |
| | | FLIR Target Positioning | |
| | Stores Management | | P-131 |
| System Management | Display Management | HUD Formatting | P-71 |
| | | VSD Formatting | P-72 |
| | | VSD Refresh | P-73 |
| | | HSD Formatting | P-74 |
| | | HSD Refresh | P-75 |
| | Aircraft State Vector Management | | P-81 |
| | Electrical Power Management | | |
| | Control Console Management | | |
| Vehicle Defense | EW/ECM | Pulseword Sorting/ Averaging/PRI | P-91, 95, 96, 97 |
| | | Emitter Classification/ Location | P-92 |
| | | Emitter Prioritization/ Correlation and Display Logic | P-93 |
| | | RHAW Receiver/Jammer Control | P-94 |
| Landing Aids | MLS | | |

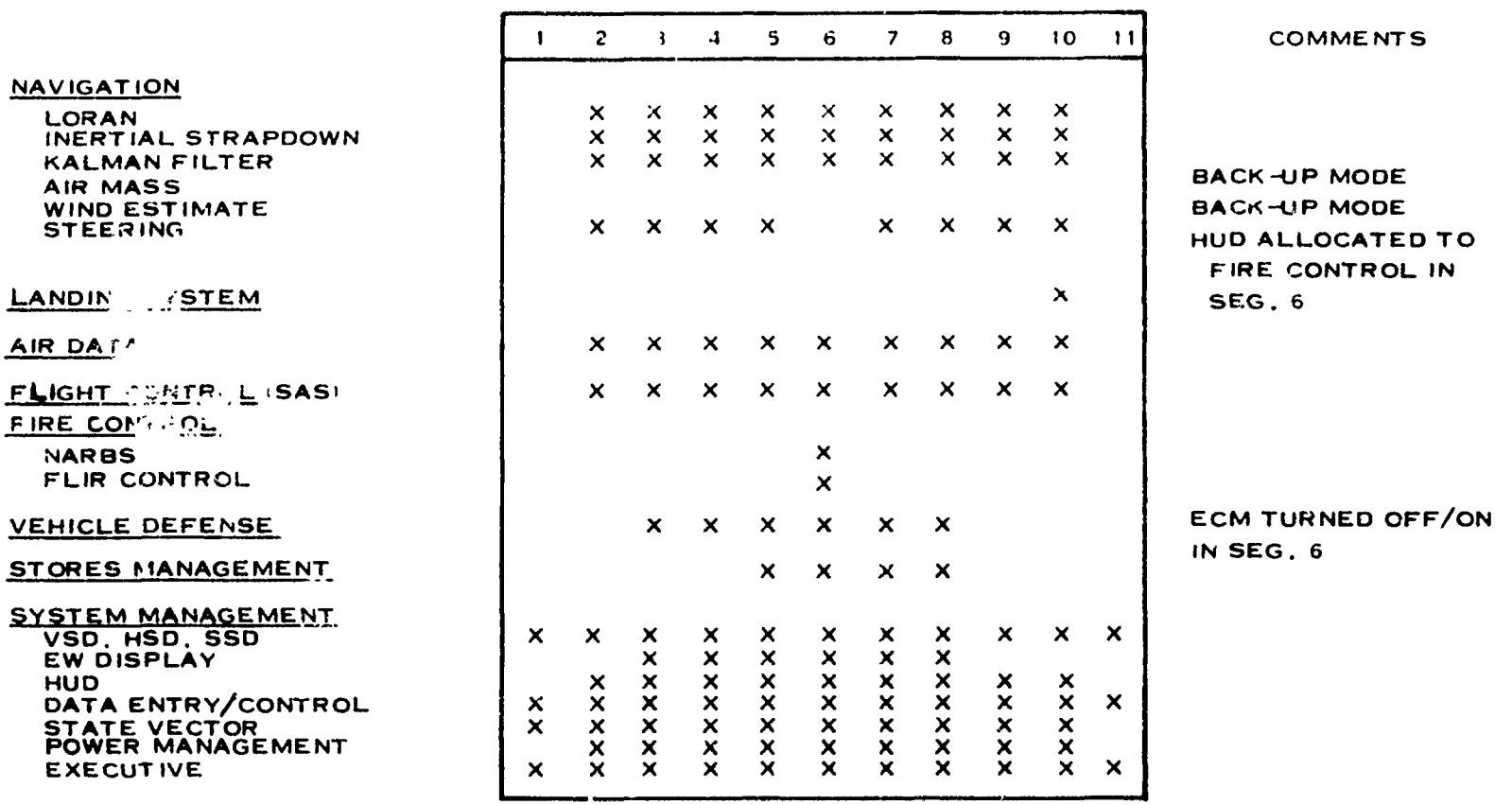



Figure 105. DP/M Test Mission Time Line

For the purpose of testing, a hypothetical test mission time-line analysis was formed showing the active mission segments for each subfunction in the system. This time-line had the following 11 mission segments:

(1) Preflight

(2) Takeoff

(3) Climb

(4) Cruise

(5) Arm Weapons/Activate ECM

(6) Attack and Weapon Delivery

(7) Evade and Depart

(8) Cruise

(9) Descend

(10) Land

(11) Postflight.

The pilot serves as the system manager during the mission, and as such is responsible for initiating either master modes of operation (e.g., select one button and cause activation of multiple subfunctions) or activation of individual subfunctions. The time-line analysis showing active mission subfunction per mission segment is shown in Figure 105.

## 7. Data Analysis and Recording Example

To illustrate the process of gathering and recording subfunction data for use either as input to the SNS or process construction analysis, the Loran avionic algorithm will be discussed. Initially, the Loran algorithm was analyzed and divided into five basic tasks. Each task is then described in terms of its scheduling requirements, inputs, computational requirements, and outputs. This type of information is represented in Table 25. The next step is to establish the interrelationship of the tasks to one another in the form of a directed graph. The directed graph shows necessary predecessor conditions that must occur prior to task initiation, as well as information (message) flow between tasks. The general convention for directed graph representations of a task is illustrated in Figure 106. The directed graph representation of the given Loran function is shown in Figure 107.

Once the information concerning a subfunction and its tasks is recorded, the data can be prepared for input to the simulator. The simulator builds a list structure describing the directed graph and a data base for each task. A portion of the actual input data for the Loran subfunction is shown in Table 26.

Figure 108 describes the list structure used to represent each task. One of these list structures is generated for each task.

In order to represent the receipt and transmittal of data a processor has an input queue representing data received and an output queue representing data to be transmitted over the bus network.

## TABLE 25. LORAN FUNCTION PROCESSING TASKS

| Function | Process Flow | Process ID | Process Description | Special Cond | Required Input Data: Data Item | No. Bits | Source | Rate | Computation Requirements: Memory Storage Instr | Memory Storage Data | Rate | Speed Ops/sec | Output Data: Data Item | Destin | No. Bits |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Loran Position Fix | H2-17 | 11 | Rcvr Interface | Rcvr on and Rcvr good | $t_{d_1}$ (TDA) | 24 | Rcvr | 25 | 650 | 200 | 25 | 5,000 | TDR 1 | Rcvr | 12 |
| | | | | | $t_{d_2}$ (TDB) | 24 | Rcvr | 25 | | | | | TDR 2 | Rcvr | 12 |
| | | | | | Rcvr status | ≈4 | Rcvr | 25 | | | | | TDRM | Rcvr | 12 |
| | | | | | | | | | | | | | Rcvr CMD | Rcvr | ≈4 |
| | | | | | $\Delta T'$ | 48 | P-14 | 25 | | | | | | | |
| | | | | | | | | | | | | | $\Delta'T$ | P-13 | 12 |
| | H2-17 | 12 | Brg Angle/expected Time Difference | | $\phi$ long | 24 | P-81 | 25 | 1,050 | 400 | 25 | 424,200 | sine $X_i$ | P-14,P-13 | 24 |
| | | | | | $\lambda$ lat | 24 | P-81 | 25 | | | | | cos $X_i$ | P-14,P-13 | 24 |
| | | | | | | | | | | | | | $\Delta T$ | P-13 | 12 |
| | | | | | Master position | 48 | table | | | | | | | | |
| | | | | | Slave 1 position | 48 | table | | | | | | | | |
| | | | | | Slave 2 position | 48 | table | | | | | | | | |
| | H2-17 | 13 | Time Diff to Lat Long | | $\Delta'T$ | 12 | P-11 | 25 | 400 | 400 | 25 | 42,050 | $\phi_L$ Long | P-15 | 24 |
| | | | | | | | | | | | | | $\lambda_L$ Lat | P-15 | 24 |
| | | | | | $\Delta T$ | 12 | P-12 | 25 | | | | | | | |
| | | | | | sine $X_i$ | 24 | P-12 | 25 | | | | | | | |
| | | | | | cos $X_i$ | 24 | P-12 | 25 | | | | | | | |
| | H2-17 | 14 | Delay Time Correction Velocity Info | | sine $X_i$ | 24 | P-12 | 25 | 150 | 200 | 25 | 359,050 | $\Delta T'$ | P-11 | 48 |
| | | | | | cos $X_i$ | 24 | P-12 | 25 | | | | | | | |
| | | | | | $V_x$ | 24 | P-81 | | | | | | | | |
| | | | | | $V_y$ | 24 | P-81 | | | | | | | | |
| | H2-17 | 15 | Universal Transverse Mercator/Output Display | | $\lambda_L$ | 24 | P-13 | 25 | 950 | 300 | 25 | 5,000 | $P_x$ | P-81 | 24 |
| | | | | | $\phi_L$ | 24 | P-13 | 25 | | | | | $P_y$ | P-81 | 24 |

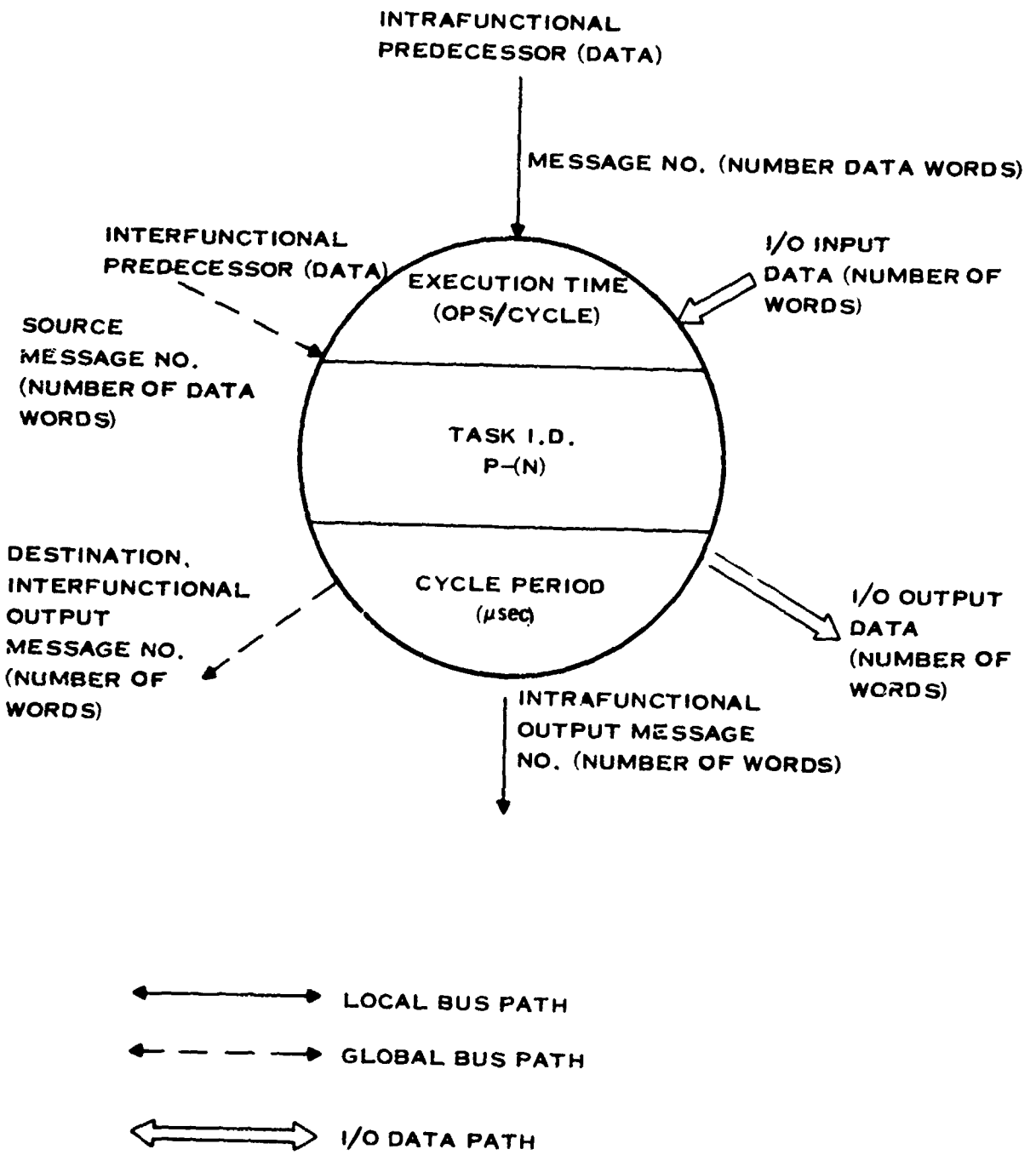



Figure 106. Directed Graph Representation Convention

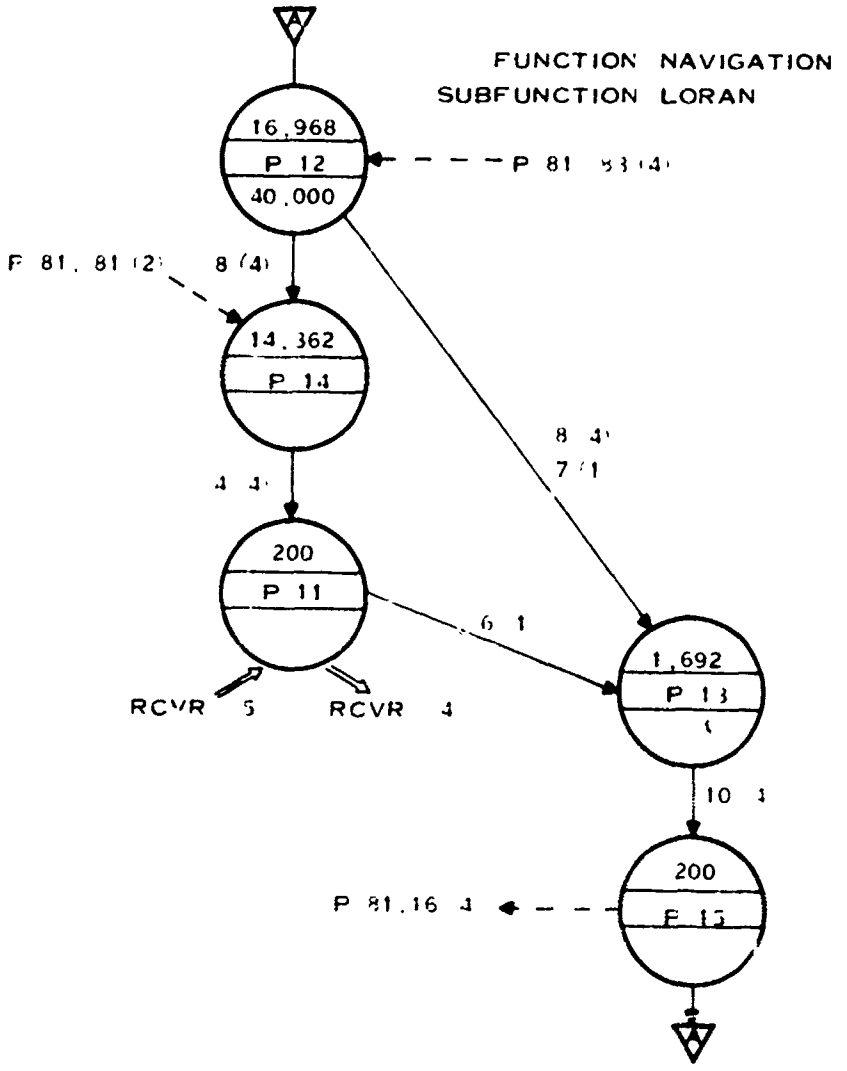



**Figure 107 Loran Directed Graph**

## 8. Virtual Message Distributor

Having described the logical precedence relationships between tasks and their pertinent parameters, one additional set of information is necessary to simulate the performance of the DP M system In particular the actual assignment of tasks to PEs (Process Constructions) and the PE interconnectivity are necessary to complete the system specification Task assignment is specified by a simple list of task identification number for each PE All of the physical characteristics of the system are utilized by the Virtual Message Distributor (VMD) to actually simulate the data flow within the DP M system Notice that the logical precedence relationships are independent of the actual method of inter-task data transmittal Only the VMD is cognizant of the method of data transmittal which is a function of the physical characteristics of the system All data delivery requests pass through the VMD which does the actual data

**TABLE 26 LORAN SNS INPUT DATA EXAMPLE***

**LORAN POSITION FIX**

```
•
•
•   &TDEEN
•   TASKID=11,
•   NPRED=1,
•   NUMSUC=1,
•   SRID=13,
•   NIMSG=1
•   IMTYPE=4
•   XTIME=200
•   TNAME=40H RCVR INTERFACE
•   &END
•
•   &MTBD
•   NUMMSG=2
•   &END
•
•   &MDEEN
•   TYPE=6,
•   LENGTH=1
•   NUMDES=1,
•   TASKID=13,
•   CCLEN=16,
•   &END
•
•   &MDEEN
•   TYPE=  1
•   LENGTH=2
•   DISTYP=1,
•   &END
•
•   &TDEEN
•   TASKID=12,
•   NPRED=1
•   NUMSUC=2
•   SRID=14,13
•   XTIME=16968
•   DELTA=40000
•   TNAME=40H BRG ANGLE EXPECTED TIME DIFFERENCE
•   &END
•
•   &MTBD
•   NUMMSG=3
•   &END
•
•   &MDEEN
•   TYPE=7,
•   LENGTH=1
•   NUMDES=1,
•   TASKID=13
•   CCLEN=16
```

*Note This data input format corresponds to a FORTRAN NAMELIST function.

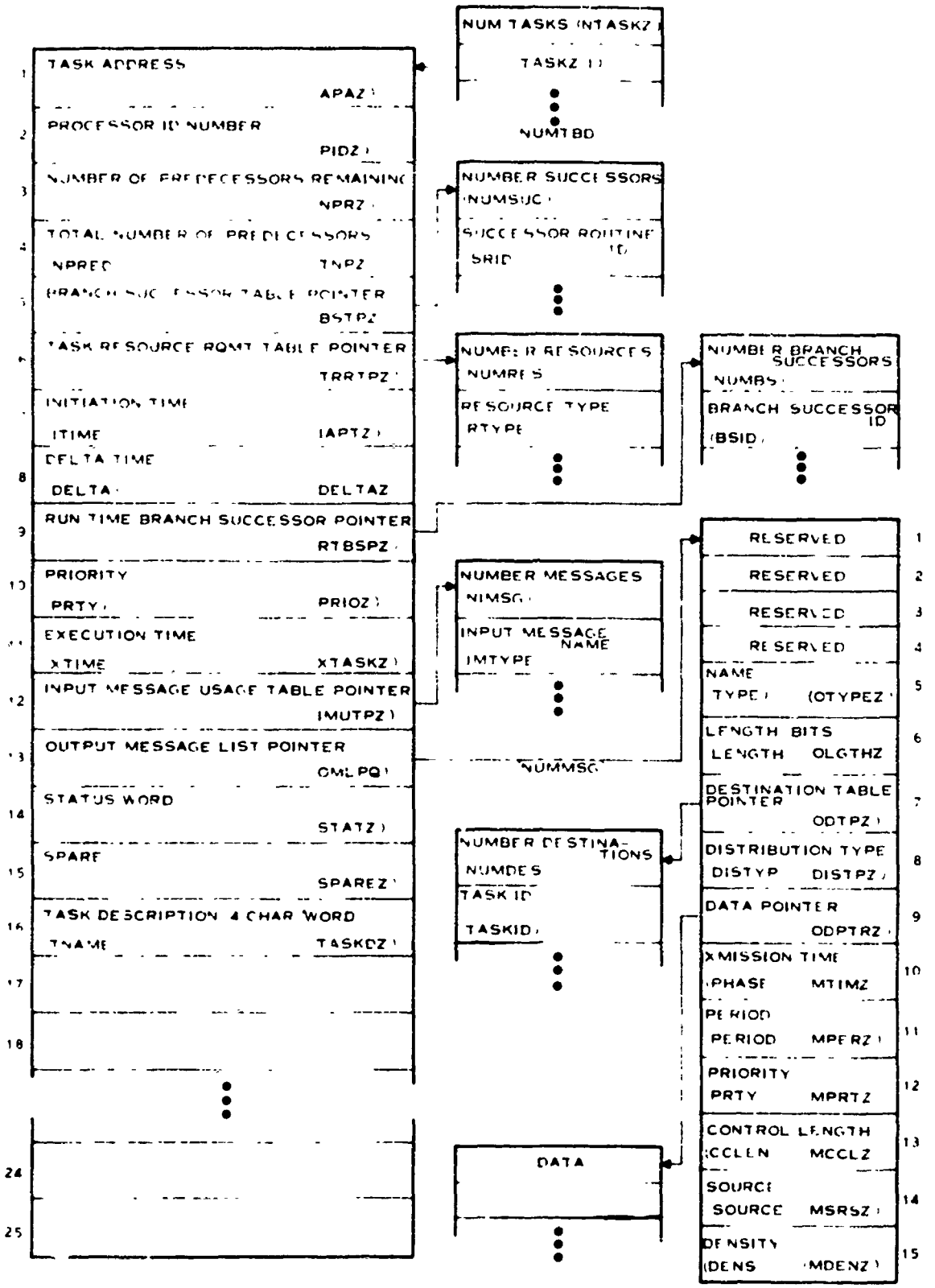



Figure 108 Task List Structure

transmission. As described previously, data is either on an input queue waiting to be used or on an output queue waiting to be transferred by the bus network. The VMD has the destination tasks for a data set via the logical precedence information as well as physical assignments. Thus, the VMD can determine if the data set must be transported over the bus network in the case where the destination task is not collocated with the source, or may be placed directly into the input queue in the case where the destination task is collocated with the source. An illustrative analogy is the case of mail delivery. A person (task) addresses an envelope (message request) containing a letter (data) to be delivered to a person (another task). The envelope is then picked up by the mailman (VMD) who determines how and where the letter should be delivered.

## 9. Model-Dependent Routines

The model-dependent routines are built using Basic Simulator routines to actually construct a simulator. To be precisely correct, model-dependent routines are all those routines that are not explicitly part of the Basic Simulator, however, there is a class of model-dependent routines that are general in nature. These routines are designed for a specific data structure organization and a specific generic class of simulation, namely distributed computer network simulation. However, these routines are not predicated on a particular model of one of the system components, i.e., bus, processors, executive, etc. These general routines allow investigation of a large class of system configurations with different components without a complete System Network Simulator redesign.

The following subsections denote these general routines developed, along with a brief description of their function and a comment on their use. Note that for a majority of the routines, a particular data structure is assumed; however, the data structure elements are referenced by name and the actual position within the structure is not important to the operation of the routine. These routines provide a general system specification mechanism as well as additional tools for simulator development.

### *a. ASSIGN (Task Assignment)*

The task assignment routine makes entries into the task definition list structure to assign avionic tasks to a particular processing element as defined by user-specified data. Figure 109 is an example of the report generated by the ASSIGN routine. This routine is used to define the avionic task configuration independently of a particular physical configuration.

### *b. DEFGEX (Define GEX Time-Ordered List)*

This routine builds the time-ordered list required by the Global Executive (GEX) scheduler

### *c. INTEND (Interrupt End)*

The interrupt end processing routine is used by an end-of-interrupt-processing event to restart the interrupted event.

### *d. INTSET (Interrupt Set)*

The interrupt process routine interrupts a currently running process if the priority of the interrupt is higher than that of the currently running process and places the interrupted process

```
**********************************************************************************

PROCESSING ELEMENT CONNECTIVITY SPECIFICATION

**********************************************************************************

GLOBAL BUS CONNECTIVITY      1     2     3     4

AFFINITY GROUP NUMBER         1 CONNECTIVITY      1     2     3

AFFINITY GROUP NUMBER         2 CONNECTIVITY      4
**********************************************************************************

AVIONIC TASKS TO PROCESSING ELEMENT ASSIGNMENT

**********************************************************************************

PROCESSING ELEMENT          1

TASKS ASSIGNED     22

PROCESSING ELEMENT          2

TASKS ASSIGNED     21    23    29    24

PROCESSING ELEMENT          3

TASKS ASSIGNED     25

PROCESSING ELEMENT          4

TASKS ASSIGNED     27    28    26
```

Figure 109. Physical Assignment Report

on a waiting queue. If the priority of the interrupt is lower than that of the currently running process, the interrupt is placed on a waiting queue. The entries in the waiting queue are removed by the INTEND routine. INTSET, in conjunction with INTEND, allows for ease of simulation of multi-level interrupt structures.

*e. READCM (Read Connectivity Matrix)*

The read connectivity routine reads the user-specified Local and Global bus connectivity definitions and builds the corresponding processing element connectivity data structures. This routine allows for flexibility in the definition of the DP/M busing structure.

*f. TESTCM (Test Connectivity Matrix)*

The test connectivity matrix routine is called by the ASSIGN routine to test for consistency between PE connectivity definition and task assignments. In addition, this routine determines whether the distribution of a message should be over the Local or Global bus and sets an indication in the message data structure.

*g. TSKDEF (Task Definition)*

The task definition routine reads the user-specified task definition data and then builds the associated list structure. Table 27 describes the input format for the task definition data. The task definition data is used to describe the logical precedence relationships between tasks and their pertinent parameters.

*h. WAKEUP*

The wakeup routine will, if the event time for a particular event is infinity (i.e., put asleep), schedule the event passed to it at a time equal to current simulation time plus a normally distributed random integer number in the interval (0,X). X is the event sample rate as specified in its definition. If the event time is not infinity, the wakeup routine performs no action. The wakeup routine provides the capability to place events in an idle state when further simulation is not required and to be "awakened" when simulation is required.

*i. XFER*

The bus transfer time routine calculates the time required to transmit a particular message over the bus. The transfer time is a function of message length in words, number of bits in the message sync, the bit period, and the gap time.

Finally, under the class of model dependent routines are those routines that explicitly model the performance of each of the particular entities in the system. These routines correspond to the starting and ending events for each entity. In fact, these routines are the DP/M System Network Simulator. These routines have been designed with as much flexibility as practically possible to provide the capability to investigate various alterations in the model. The following is a list of the model-dependent routines along with a brief discussion of their function.

## TABLE 27 SNS INPUT TASK DATA FORMAT

| Variable | Description | Default Value/ Restrictions |
|---|---|---|
| As many of the following data sets as there are tasks | | |
| &TDEFN | Start NAMELIST data set descriptor | |
| LAST | = True If last task definition | |
| TASKID | Task identification number | |
| NPRED | Number of predecessors to task | 0 |
| DELTA | Inverse of rep rate (period) | $2^{**}-1$ |
| TNAME=40H | Description of task (up to 40 characters) | Blanks |
| NUMSUC | Number of successor tasks | 0 |
| SRID | List of successor tasks id's | |
| RTYPE | 16-bit words of memory used | |
| NUMBS | Number of run time branch successors | 0 |
| BSID | List of run time branch successors | |
| NIMSG | Number of input message required by task | 0 |
| IMTYPE | List of input message types | |
| XTIME | Tasks execution time in processor clocks | 0 |
| &END | End NAMELIST data set | |
| &MTBD | Start NAMELIST data set | |
| NUMMSG | Number of output messages generated by task | |
| &END | End NAMELIST data set | |
| As many of the following NAMELIST data sets as specified by NUMMSG | | |
| &MDEFN | Start NAMELIST data set | |
| TYPE | Message type | 0 |
| LENGTH | Message length in words exclusive of overhead | 0 |
| NUMDES | Number of destination tasks | 1 |
| TASKID | List of destination tasks | |
| PHASE | Transmission time | 0 |
| PERIOD | Period of message transmission | 1 |
| PRTY | Priority of message | 0 |
| CCLEN | Control length | 0 |
| SOURCE | Source id | 0 |
| DENS | Density | 0 |
| &END | End NAMELIST data set | |

## 10. Bus Control Model Events

### *a GBUSCR (Global Bus Control Receive)*

The Global bus control receive event routine removes a message from the Global bus queue and places it on the destination processing element(s) input queue(s).

### *b GBUSCX (Global Bus Control Transmit)*

The Global bus control transmit event routine attempts to remove a message from the output queue of the processor that currently has the bus control. If a message is present, GBUSCX places it on the Global bus queue and schedules the Global bus control receive event at a later time equal to the transmission time of the message. If no message is present on the processor output queue, a sync signal is generated and the routine is rescheduled at a later time equal to a bus control pass time. In either case, bus control position is advanced to the next processing element in the loop. The model developed also allows for supercommutation of one or more processors (i.e., a processor may gain access to the bus at a higher frequency than other processors).

To increase simulator efficiency with no loss in information or simulation accuracy, the Global bus control transmit event places itself into an idle state if no messages are delivered during a complete pass around the bus. Any event that generates an output message will "wakeup" the bus control event. At that point, the actual simulation of the bus performance, in terms of transferring messages, passing of control, etc., continues as if the event had never been idle.

### *c LBUSCR (Local Bus Control Receive), LBUSCX (Local Bus Control Transmit)*

The Local bus control receive (transmit) event routine functions exactly as does the Global bus control receive (transmit) event except that it models the local buses in the system.

## 11. Avionic Task Model Events

Each avionic task is modeled by two events, the task start event and the task end event. This is necessary to be able to simulate the total execution time for the task. The Local Executive (LEX) model transfers control to the task start event when the conditions are proper. The task start event schedules the end event at a time equal to now plus the execution time of the task. This simulates total task time. The end event saves the task ID, status, and branch successor identification if any and schedules the output message request module, thus returning control to the LEX.

## 12. Executive Model Events

The executive model events exactly follow the logical decisions made by the Local and Global Executives. From the simulation standpoint, the executive events are no different from any other event within the system.

*a. INTTIM (Interval Timer)*

The interval timer event simulates a GEX programmable interval timer interrupt and calls the GEX time scheduler.

*b GEXTIM (GEX Time Scheduler)*

The GEX time scheduler event schedules all subfunctions in the system. It uses and maintains a time-ordered list of subfunction start times. It also monitors the active, abort, and complete flags associated with subfunctions.

*c GEXEND (GEX Time Scheduler End)*

The GEX time scheduler end event schedules the programmable interval timer event based upon GEXTIM computations

*d IMSG (Input Message Scanner)*

The input message scanner for the LEX event routine processes messages transmitted over the Local or Global buses. It provides association between messages and recipients and requests data moves to the dispatcher if necessary. It also incorporates a model for receiving activate/deactivate and completion command messages from subfunctions.

*e DECMNT (Decrement Predecessor Count)*

This routine is a LEX event that decrements predecessor counts of subfunctions.

*f. OMSG (Output Message Interpreter)*

The output message interpreter is a LEX event that processes output message requests and places them on the processor output queue.

*g SCHED (LEX Task Complete)*

This event routine performs the task-complete processing for the LEX.

*h BSSCAN (Branch Successor Scan)*

This event routine scans the branch successor list and decrements the predecessor count if the task supplies a branch successor ID.

*i DISPTN (Dispatcher)*

This routine is a service module called by several submodules to reset flags and move ready tasks to the ready queue.

*j SSCAN (Successor Scan)*

This event routine examines the preamble of the completing task's successors and decrements predecessor counts

### k. *IMSCAN (Input Message Scan)*

The input message scanner for the GEX event routine processes messages transmitted over the Local or Global buses. It provides association between messages and recipients and requests moves to the dispatcher if necessary.

### l. *OMRQST (Output Message Request)*

The output message request interpreter is a GEX event routine that processes message requests from tasks and prepares them for output by placing them on an output queue.

## 13. Data Collection and Report Generation

A family of data-collection and report-generation programs was developed for the System Network Simulator. These programs provide the capability to selectively collect data on and generate reports for the various system parameters under investigation for a particular DP/M configuration and/or avionic mission segment. Both the collection and dispensation of data as well as the generation of reports are controlled by user specified parameters. In general, the user has four options: (1) no data is collected and no reports generated; (2) data is collected and saved, but no report generated; (3) data is collected and report generated but data not saved; (4) data is collected and saved, and reports generated. Data saved on magnetic tape may be processed at a later time. In fact, this saved data may be used at a later time to compare results of two or more different simulation experiments. However, care must be exercised when comparing results generated using random numbers. The particular options available are discussed in detail in the following subsections.

Data collection and report generation occur at three distinct levels: (1) event level, (2) sample period level, (3) postsimulation level. At each of these levels, reports concerning bus performance, processor loading, executive performance, and number of avionic tasks processed may be selectively generated.

### a. *Event-Level Reports*

Event-level reports consist of those reports that are generated at the completion of each simulation event. An event-level report provides a single-line output that indicates the current event executing. These reports are used to provide an event-by-event trace of the flow of the simulation throughout a particular simulation experiment. Figure 110 is an example of an event-level-report format. "NAME" is the name of the event routine currently executing. "NOW" is current simulation time. "N" is the identification of the entity currently being processed by the event. N is normally an avionic task or message identification number.

```
** EVENT ROUTINE 'NAME' AT TIME 'NOW' PROCESSING 'N'
```

**Figure 110. Event-Level Report Format**

```
**********************************************************************
DPM SIMULATION TEST CASE                                  12:18:40   05/28/74
BUS ACTIVITY RECORD FOR BUS NUMBER          1
MAJOR FRAME NUMBER    1                                MINOR FRAME NUMBER    1
----------------------------------------------------------------------
MINOR FRAME START TIME          =     0.0
TOTAL BUS USAGE TIME            =     0.000276
PERCENT OF MINOR FRAME UTILIZED =     5.52
----------------------------------------------------------------------
NUMBER OF MESSAGES TRANSMITTED  =     5
MAXIMUM TIME ON OUTPUT QUEUE    =     0.000164          FOR MESSAGE        400
MINIMUM TIME ON OUTPUT QUEUE    =     0.000012          FOR MESSAGE         22
AVERAGE TIME ON OUTPUT QUEUE    =     0.000080
LENGTH OF LONGEST MESSAGE TRANSMITTED  =          70
LENGTH OF SHORTEST MESSAGE TRANSMITTED =          37
AVERAGE LENGTH OF MESSAGE TRANSMITTED  =       50.00
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER        22
MESSAGE LENGTH =        70     ORIGIN =       1     DESTINATION =            2
RELATIVE TIME OF START =     0.000312    LENGTH OF TRANSMISSION =      0.000075
TIME ON OUTPUT QUEUE   =     0.000012
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER       400
MESSAGE LENGTH =        37     ORIGIN =       1     DESTINATION =            2
RELATIVE TIME OF START =     0.000403    LENGTH OF TRANSMISSION =      0.000042
TIME ON OUTPUT QUEUE   =     0.000103
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER        24
MESSAGE LENGTH =        70     ORIGIN =       2     DESTINATION =            3
RELATIVE TIME OF START =     0.002735    LENGTH OF TRANSMISSION =      0.000075
TIME ON OUTPUT QUEUE   =     0.000015
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER       400
MESSAGE LENGTH =        37     ORIGIN =       2     DESTINATION =            1
RELATIVE TIME OF START =     0.002826    LENGTH OF TRANSMISSION =      0.000042
TIME ON OUTPUT QUEUE   =     0.000106
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER       400
MESSAGE LENGTH =        37     ORIGIN =       2     DESTINATION =            3
RELATIVE TIME OF START =     0.002884    LENGTH OF TRANSMISSION =      0.000042
TIME ON OUTPUT QUEUE   =     0.000164
**********************************************************************
```

Figure 111 Sample Period Bus Report

### *b. Sample Period Reports*

Sample period reports consist of those reports that are generated at the completion of a user-specified sample time. Data are collected during a period of simulation time (sample period) and then reports (if specified) are generated at the completion of that time. Figure 111 is an example of a sample period report for a local bus. Figure 112 is an example of a sample period report for a processor. The user may specify which reports are to be generated.

In all report examples, times are measured in seconds. A minor frame is the user-defined sample period. A major frame is a user-defined sample period that contains an integer number of minor frames. Message numbers are those identifications assigned by the system test data. Message lengths are measured in bits including all overhead bits. Origin and destination are the message origin and destination processor identification number. Relative time of start is measured from the start of the minor frame and bus usage is measured as a percentage of total available bus bandwidth

### *c. Post-Simulation Reports*

Post-simulation reports consist of those summary type reports that are generated after the completion of a particular simulation experiment. Post-simulation reports provide a concise summary of all data collected throughout the simulation. These summaries allow rapid evaluation of simulation results without massive data reduction. If, after initial evaluation, it is necessary to investigate in further detail, sample period or even event level reports can be used. The user may specify explicitly which reports are to be generated, even though data was collected on all entities. Figure 113 is an example of the bus decomposition report. The bus decomposition report shows the relative usage of the bus bandwidth by each component of a message. Figure 114 is an example of the bus loading bar graph. This graph shows the percentage of bus usage during each sample period. Figure 115 is an example of the bus usage summary report. This report summarizes the performance of each bus in the DP/M system during the complete simulation experiment. This report is actually a summary of the sample period bus activity reports. Also associated with the bus usage summary is a summary of all messages transferred over that particular bus. Figure 116 is an example of the message usage summary report. Reports for each bus may be selected by the user. Figure 117 is an example of the processor usage bar graph. This graph shows the percentage of processor usage by interrupt servicing, system (executive) programs, and applications programs during each sample period. Figure 118 is an example of the processor usage summary report. This report summarizes the usage of each processor in the DP/M system during the simulation experiment. This report is actually a summary of the sample period processor usage reports.

## E. PROCESSING ELEMENT SIMULATION

The objective of the Processing Element Simulator is to evaluate the major hardware design alternatives of a PE. In particular, the Processing Element Simulator is used to analyze and evaluate: (1) PE operations with respect to internal program execution and external inputs; (2) major hardware design considerations and alternatives of the PE. As such, the PE Simulator must simulate the internal performance of the PE to a much finer detail than was required by the System Network Simulator. However, a new simulation concept is not required by the PE Simulation: it is, in fact, just a logical extension of the SNS to the next level of detail (i.e., the discrete event-oriented simulation control structure is retained by the PE Simulator).

```
**********************************************************************
DPM SIMULATION TEST CASE                                  12:18:40   05/28/74
COMPUTER UTILIZATION FOR PROCESSOR 2
----------------------------------------------------------------------

MAJOR FRAME NUMBER =     1                  MINOR FRAME NUMBER =     1
REPORT START TIME  =     0.0                REPORT STOP TIME   =     0.005000
----------------------------------------------------------------------

                          TIME UTILIZATION DATA
NUMBER OF INTERRUPTS SERVICED   =          0
TOTAL INTERRUPT SERVICE TIME    =    0.0                  PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =    0.000300             PERCENT =    6.00
TOTAL APPLICATIONS PROGRAM TIME =    0.002270             PERCENT =   45.40
TOTAL IDLE TIME                 =    0.002430             PERCENT =   48.60
----------------------------------------------------------------------

                          MEMORY UTILIZATION DATA
AVAILABLE MEMORY =      4000
MAXIMUM USED     =         0                        AVERAGE USED =         0
----------------------------------------------------------------------

                          MESSAGE UTILIZATION DATA
NUMBER OF INCOMING MESSAGES =         3     INCOMING MESSAGES MISSED =         0
NUMBER OF OUTGOING MESSAGES =         3     OUTGOING MESSAGES MISSED =         0
**********************************************************************
```

Figure 112. Sample Period Processor Report

```
**************************************************************************
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS    1
--------------------------------------------------------------------------
TOTAL TRAFFIC                 =      20.96 KBPS
TRAFFIC UTILIZED FOR DATA     =      15.60 KBPS   74.41 PERCENT
TRAFFIC UTILIZED FOR HEADER   =       2.26 KBPS   10.77 PERCENT
TRAFFIC UTILIZED FOR SYNC     =       1.16 KBPS    5.56 PERCENT
TRAFFIC UTILIZED FOR GAP      =       1.94 KBPS    9.26 PERCENT
**************************************************************************
**************************************************************************
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS    2
--------------------------------------------------------------------------
TOTAL TRAFFIC                 =       0.0  KBPS
**************************************************************************
**************************************************************************
                BUS TRAFFIC DECOMPOSITION FOR GLOBAL BUS   99
--------------------------------------------------------------------------
TOTAL TRAFFIC                 =      25.24 KBPS
TRAFFIC UTILIZED FOR DATA     =      22.60 KBPS   89.56 PERCENT
TRAFFIC UTILIZED FOR HEADER   =       1.51 KBPS    5.97 PERCENT
TRAFFIC UTILIZED FOR SYNC     =       0.42 KBPS    1.68 PERCENT
TRAFFIC UTILIZED FOR GAP      =       0.71 KBPS    2.80 PERCENT
**************************************************************************
```

Figure 113. Bus Decomposition Report

```
DPM SIMULATION TEST CASE                                    10:45:35    06/03/74
.  FRAME        BUS ACTIVITY SUMMARY REPORT FOR BUS   1                PAGE     1
. MAJ MIN       TIME    PERCENT UTILIZATION GRAPH                              .
-------------------------0--------20--------40--------60--------80-------100
.   1   1     0.000000   5.52.BBB  .         .         .         .         .
.   1   2     0.005000   0.00.B    .         .         .         .         .
.   1   3     0.010000   3.18.BB   .         .         .         .         .
.   1   4     0.015000   0.00.B    .         .         .         .         .
.   1   5     0.020000   3.18.BB   .         .         .         .         .
.   1   6     0.025000   0.00.B    .         .         .         .         .
.   1   7     0.030000   3.18.BB   .         .         .         .         .
.   1   8     0.035000   0.00.B    .         .         .         .         .
.   1   9     0.040000   5.52.BBB  .         .         .         .         .
.   1  10     0.045000   0.00.B    .         .         .         .         .
-------------------------0--------20--------40--------60--------80-------100
```

Figure 114. Bus Loading Bar Graph

```
DPM SIMULATION TEST CASE                                    12:18:40    05/28/74
                    BUS UTILIZATION SUMMARY FOR BUS    1                PAGE     1

REPORT START TIME =     0.0                          REPORT STOP TIME =     0.085000

TOTAL BUS USAGE TIME      =     0.001782
AVERAGE BUS UTILIZATION   =     2.10
MAXIMUM BUS UTILIZATION   =     5.52      DURING MAJOR FRAME     1 MINOR FRAME     1
MINIMUM BUS UTILIZATION   =     0.0       DURING MAJOR FRAME     1 MINOR FRAME     2

TOTAL NUMBER OF MESSAGES TRANSMITTED  =          33
MAXIMUM TIME ON OUTPUT QUEUE          =    0.000164     FOR MESSAGE          400
                                       DURING MAJOR FRAME     1 MINOR FRAME     1
MINIMUM TIME ON OUTPUT QUEUE          =    0.0          FOR MESSAGE           24
                                       DURING MAJOR FRAME     2 MINOR FRAME     7
AVERAGE TIME ON OUTPUT QUEUE          =    0.000055
LENGTH OF LONGEST MESSAGE TRANSMITTED =          70
                                       DURING MAJOR FRAME     1 MINOR FRAME     1
LENGTH OF SHORTEST MESSAGE TRANSMITTED =         37
                                       DURING MAJOR FRAME     1 MINOR FRAME     1
AVERAGE LENGTH OF MESSAGE TRANSMITTED =        49.00
```

Figure 115. Bus Usage Summary Report

```
DPM SIMULATION TEST CASE                                    12:18:40    05/28/74
.                MESSAGE TRANSMISSION SUMMARY FOR BUS    1          PAGE     2
-------------------------------------------------------------------------------
REPORT START TIME =     0.0                        REPORT STOP TIME =     0.085000
-------------------------------------------------------------------------------
MESSAGE SUMMARY FOR MESSAGE NUMBER        22                                   .
NUMBER OF TIMES MESSAGE TRANSMITTED =      9                                   .
MESSAGE LENGTH =     70       LENGTH OF TRANSMISSION =     0.000075          .
ORIGIN          =     1                                                       .
DESTINATION     =     2                                                       .
MAXIMUM TIME ON OUTPUT QUEUE =    0.000016                                    .
.                                 DURING MAJOR FRAME    2 MINOR FRAME    3
MINIMUM TIME ON OUTPUT QUEUE =    0.000006                                    .
.                                 DURING MAJOR FRAME    1 MINOR FRAME    3
AVERAGE TIME ON OUTPUT QUEUE =    0.000010                                    .
*******************************************************************************
MESSAGE SUMMARY FOR MESSAGE NUMBER        24                                   .
NUMBER OF TIMES MESSAGE TRANSMITTED =      3                                   .
MESSAGE LENGTH =     70       LENGTH OF TRANSMISSION =     0.000075          .
ORIGIN          =     2                                                       .
DESTINATION     =     3                                                       .
MAXIMUM TIME ON OUTPUT QUEUE =    0.000015                                    .
.                                 DURING MAJOR FRAME    1 MINOR FRAME    1
MINIMUM TIME ON OUTPUT QUEUE =    0.0                                         .
.                                 DURING MAJOR FRAME    2 MINOR FRAME    7
AVERAGE TIME ON OUTPUT QUEUE =    0.000006                                    .
*******************************************************************************
MESSAGE SUMMARY FOR MESSAGE NUMBER       400                                   .
NUMBER OF TIMES MESSAGE TRANSMITTED =     21                                   .
MESSAGE LENGTH =     37       LENGTH OF TRANSMISSION =     0.000042          .
ORIGIN          =     1    2                                                  .
DESTINATION     =     2    1    3                                             .
MAXIMUM TIME ON OUTPUT QUEUE =    0.000164                                    .
.                                 DURING MAJOR FRAME    1 MINOR FRAME    1
MINIMUM TIME ON OUTPUT QUEUE =    0.000003                                    .
.                                 DURING MAJOR FRAME    2 MINOR FRAME    1
AVERAGE TIME ON OUTPUT QUEUE =    0.000082                                    .
*******************************************************************************
```

Figure 116. Message Transmission Summary Report

```
DPM SIMULATION TEST CASE                                          12:18:40    05/28/74
.  FRAME          COMPUTER UTILIZATION REPORT FOR PROCESSOR 2                 PAGE     1
. MAJ MIN          TIME    PERCENT UTILIZATION GRAPH                                  .
                                  0--------20--------40--------60--------80-------100
.   1   1      0.000000  51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1   2      0.005000   0.00.          .         .       .         .         .
.   1   3      0.010000  51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1   4      0.015000   0.00.          .         .       .         .         .
.   1   5      0.020000  51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1   6      0.025000   0.00.          .         .       .         .         .
.   1   7      0.030000  51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1   8      0.035000   0.00.          .         .       .         .         .
.   1   9      0.040000  51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1  10      0.045000   0.00.          .         .       .         .         .
                                  0--------20--------40--------60--------80-------100
.    I = INTERRUPT SERVICE     S = SYSTEMS PROGRAM     A = APPLICATIONS PROGRAM    .
```

Figure 117. Processor Usage Bar Graph

```
DPM SIMULATION TEST CASE                                    12:18:40    05/28/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 2              PAGE     1

REPORT START TIME =      0.0                          REPORT STOP TIME =      0.085000

.                              TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED    =              0                                  .
TOTAL INTERRUPT SERVICE TIME     =      0.0                      PERCENT =     0.0
TOTAL SYSTEMS PROGRAM TIME       =      0.002700                 PERCENT =     3.18
TOTAL APPLICATIONS PROGRAM TIME  =      0.020430                 PERCENT =    24.04
TOTAL IDLE TIME                  =      0.061870                 PERCENT =    72.79

.                              MEMORY UTILIZATION DATA                            .
AVAILABLE MEMORY =       4000                                                     .
MAXIMUM USED     =          0                          AVERAGE USED =           0

.                              MESSAGE UTILIZATION DATA                           .
NUMBER OF INCOMING MESSAGES =          29     INCOMING MESSAGES MISSED =          0
NUMBER OF OUTGOING MESSAGES =          21     OUTGOING MESSAGES MISSED =          0
```

Figure 118. Processor Usage Summary Report

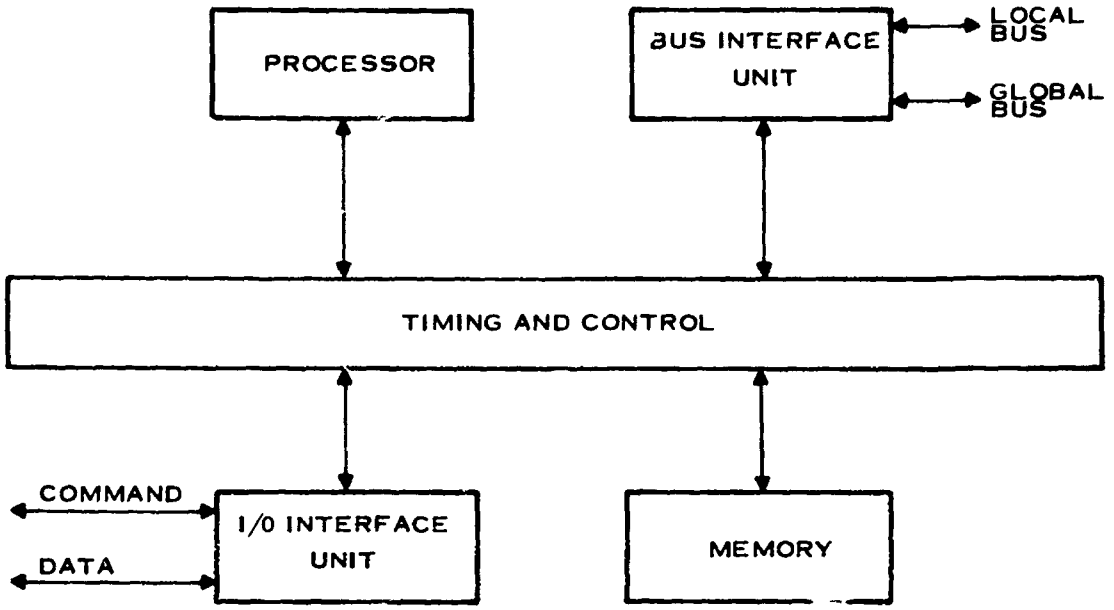



**Figure 119. PE Block Diagram**

## 1. Processing Element Simulator Organization

The PE Simulator is made up of five distinct functional models: the processor, the memory, the I/O interface, the bus interface, and the intra-processor element timing and control. Figure 119 is a block diagram of the PE as viewed by the PE Simulator.

Each of the event models that describe these functions is scheduled to occur at some future time by placing it on the simulation Future Event List (FEL). The FEL is a chronologically ordered list that contains all future events. Each event is then simulated in its order of occurrence in time. Interaction, resource conflict resolution, and sequencing of events is automatically controlled by the simulator structure and takes place during the execution of the simulation experiment. Figure 120 is an example of the sequencing of intra-PE events on the FEL. The instruction execution event simulates the execution of one instruction and schedules itself at a future time equal to the length of execution of the instruction. Any Direct Memory Access (DMA) transfer event executes next, effecting a transfer to/from memory, and then schedules itself at a future time equal to the DMA transfer rate. The interrupt event causes an interrupt to be generated and may be rescheduled, depending on the interrupt type. These events continue to be posted on the FEL, removed for execution, posted on the FEL, and on throughout the simulation. It is important to realize that it is not necessary to be able to predict the order of occurrence of events *a priori*.

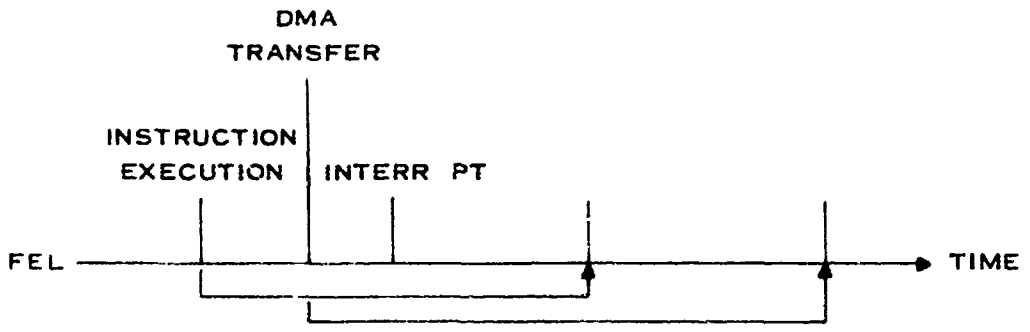



Figure 120. PE Event Sequencing

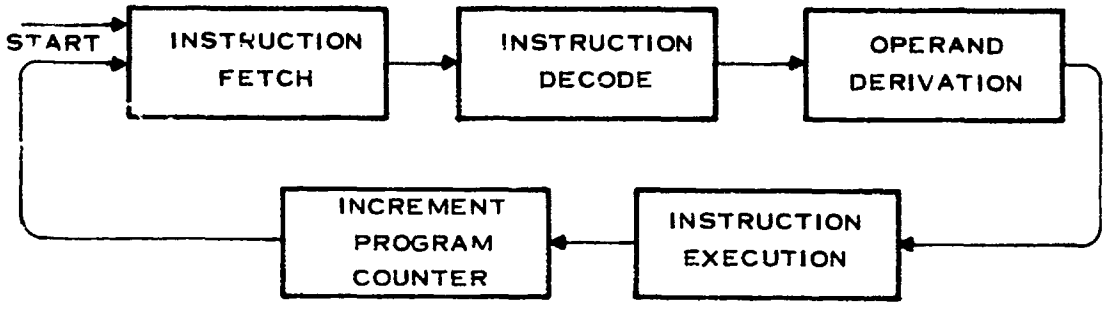



Figure 121. Instruction Execution Cycle Block Diagram

## 2. Processor Simulation

The processor model describes the various machine registers available to the programmer through the instruction set the execution operation of each processor instruction, the timing of each execution operation, and the internal processor interrupt structure. The set of simulation routines that simulate the processor will hereafter be referred to as the Instruction Level Simulator (ILS).

### *a. ILS Inputs*

Binary program object code modules generated by the assembler representing either avionic application functions or selected test code kernels are inputs to the ILS. The binary object modules are link-edited together by the PE Simulator monitor to form an absolute object module which is loading into a pseudo-memory that represents the actual processor memory. Variables that describe processor performance parameters such as clock speed, memory read, write times, memory size, etc., are located in a single FORTRAN COMMON block to facilitate modification. To begin simulation, the program counter is initialized to the hardware "GO" location and the first instruction is fetched from pseudo-memory for simulation.

### *b ILS Operations*

The ILS functional operation emulates the actual processor hardware. The instruction execution sequence is represented in Figure 121. Table 28 is a list of all processor instruction algorithms. Each step in instruction execution simulation, as well as description of some of the FORTRAN routines, will be described in detail in the following subsections.

### *c Instruction Execution Event*

The instruction execution event (IEODEX) is the discrete event that represents the occurrence of a processor instruction execution. The actual instruction simulation takes place in the instruction execution simulation routine (SMEIR). SMEIR is described in the following

**TABLE 28 PROCESSOR INSTRUCTION SIMULATION ALGORITHMS**

Arithmetic

- LOAD TWO'S COMPLEMENT
- ADD
- SUBTRACT
- COMPARE SIGNED
- MULTIPLY
- DIVIDE
- ADD CONSTANT SHORT
- LOAD TWO'S COMPLEMENT DOUBLE
- ADD DOUBLE
- SUBTRACT DOUBLE
- COMPARE DOUBLE

Program and Interrupt Control

- BRANCH ON CONDITION
- EXCHANGE STATUS AND PC
- RETURN FROM INTERRUPT
- COMPARE CONSTANT SHORT
- BRANCH ON CONDITION SHORT
- BRANCH INDIRECT AND LINK REGISTER SHORT
- INCREMENT AND BRANCH IF NEGATIVE SHORT
- LOAD STATUS
- COMPARE UNSIGNED
- DECREMENT AND BRANCH IF POSITIVE
- SET STATUS

Bit

- SET BIT UPPER BYTE
- SET BIT LOWER BYTE
- CLEAR BIT UPPER BYTE
- CLEAR BIT LOWER BYTE
- TEST BIT UPPER BYTE
- TEST BIT LOWER BYTE

Data Transfer

- LOAD
- STORE
- STORE THROUGH MASK
- PUSH
- MOVE AND AUTO INCREMENT
- PUSH MULTIPLE
- POP MULTIPLE
- LOAD DIRECT SHORT
- STORE DIRECT SHORT
- LOAD CONSTANT SHORT
- READ STATUS
- LOAD DOUBLE
- STORE DOUBLE

Logical

- LOAD ONE'S COMPLEMENT
- AND
- OR
- EXCLUSIVE OR
- AND TO MEMORY
- OR TO MEMORY

Shift

- SHIFT SINGLE
- SHIFT DOUBLE

Input Output

- REGISTER INPUT
- REGISTER OUTPUT
- MEMORY INPUT
- MEMORY OUTPUT
- PSEUDO I/O

subsection. IFODEX controls any concurrency or conflicts with other PE model events. This routine verifies that no higher priority events are pending at the current time and then calls the instruction simulation routine. At the completion of the simulation routine, IFODEX checks if the next event on the FEL is due to occur before the next instruction. If there is an event pending, the IFODEX event is placed on the FEL. If there is no event pending, simulation time is advanced, and the simulation routine is called to simulate the next instruction. This look-ahead eliminates the unnecessary placing and removing of the IFODEX routine on the FEL. Since the majority of the PE simulation time is spent during instruction simulation, this minor test results in a major decrease in simulation run time.

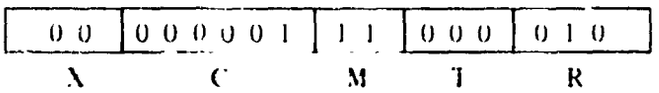
### *d Instruction Simulation Routine*

The main instruction simulation routine (SMLTR) contains the logic to simulate instruction fetch, operand derivation, and execution. The first step in instruction simulation is to fetch from pseudo-memory the instruction to be simulated. The instruction word is then passed to the instruction decode routine (IDEC). IDEC extracts the various fields from the instruction and sets a flag (TYPE) that indicates the source format type. Figure 122 shows the various values of the TYPE flag. Next, the various fields of the instruction are passed to either the long or short instruction operand derivation routine (LOPDER or SOPDER), depending on the type instruction format. The LOPDER and SOPDER routines compute the derived operand for the instruction based on the modification field value. Control is then passed to the execution simulation section by use of a FORTRAN-computed GO TO based upon the value of the instruction opcode. Each instruction execution algorithm shares a common data base of variables that contains such things as processor register file, status word, value of derived operand, etc. As

| TYPE | SOURCE FORMAT | | KEY |
|---|---|---|---|
| 1 | op R | r,t | X=0, M=0 |
| 2 | op C | r,A | X=0, M=2, t=7 |
| 3 | op R.I | r,t | X=0, M=1 |
| 4 | op R.IA | r,t | X=0, M=2, t≠7 |
| 5 | op.D | r,A | X=0, M=3, t=0 |
| 6 | op.DX | r,A,t | X=0, M=3, t≠0 |
| 7 | op | CAW,A | X=1, C=6 or 7, r=0 |
| 8 | op | r,A | X=1 |
| 9 | LDS | r,D,t | X=2 |
| 10 | SDS | r,D,t | X=3 |
| 11 | op | CAW,A,r | X=1, C=7 or 8, r≠1 |
| 12 | op C | r,A,A' | X=0, M=2, t=7, Double Instruction |

Figure 122. Instruction Format Types

stated previously, a particular algorithm is selected based upon the instruction opcode and execution is performed. At the completion of execution simulation, control is returned to IFODEX. As an example, consider a load instruction with the direct modification which has the following binary format:




First, IDEC will determine that the instruction is of the standard format. Knowing the format, IDEC extracts the various fields and stores the following variables into the common data base: X = 0, C = 1, M = 3, T = 0, R = 2, TYPE = 5. Since the instruction is of the standard format, LOPDER is called to compute the derived operand (DO). LOPDER uses the value M to determine that the instruction is a direct modification. LOPDER fetches the A-field (next word) and increments the PC. Finally, LOPDER returns the contents of the address pointed to by the A-field as the DO. The value of C is then used in a computed GO TO to pass control to the load simulation algorithm. The load simulation algorithm loads the DO into the register specified by the value of R. Lastly, the print line is formatted and control is returned to IFODEX.

The execution time for the particular instruction is computed by use of a table lookup based upon its operation code. Any additional time required for the operand derivation and special execution time conditions is also added. Special execution time conditions include such things as the increased time when a branch is taken on a condition branch instruction.

### *e. Interrupt Simulation*

The simulation of the interrupt structure parallels the following hardware characteristics: (1) a stimulus to an armed or enabled interrupt will cause a trap to be simulated using the data address specified by the originator; (2) a stimulus to an unarmed interrupt will be remembered; (3) stimuli to two armed interrupts will result in the interrupt of the highest hardware priority trapping first; (4) an interrupt stimulus that occurs during the execution of an instruction will occur after completion of instruction execution. A diagnostic warning message is printed if an interrupt stimulus was present but could not trap since the interrupt was not armed. The simulation of the interrupt structure of the computer allows the following options in creating interrupt stimuli to a program by means of parameter cards:

specification of the interrupt priority and corresponding trap address

specification of the frequency of occurrence of that interrupt

specification of a time interval to be used in the frequency of occurrence.

In addition to assigning the hardware priority of an interrupt stimulus, it is also possible to specify the frequency and interval of time at which the interrupt stimulus is to occur. There are four possible frequency options that may be defined

R interrupt to occur at random time intervals

F interrupt to occur at a fixed time interval

S interrupt to occur at single time interval (i.e., only once during the simulation)

C interrupt to occur a some interval of time after the issuing of a specified CAW

Each of the above frequencies can be associated with both internal and external interrupt stimuli. Since there is no actual distinction made between an internal or external interrupt request, the simulator does not require such a distinction. It should also be noted that frequency type C is used to form the amount of time required to acknowledge a programmed controlled I/O transfer.

The actual time interval between simulator-generated interrupts is defined in the form of a time-to-interrupt equation. This interrupt-schedule equation enables the simulator to calculate the time the next interrupt will occur. The equation field of the parameter definition card is divided into a mantissa and characteristic to provide a wide range of interrupt time intervals. The form of the equation on the parameter card is

mmmcc

where

mmm = the mantissa

cc = the characteristic.

The above field can be translated into the following equation

$$mmm \times 10^{-cc} \text{ seconds}$$

Interrupt simulation consists of an interrupt event routine (INTENT), an interrupt stimulus generation routine (INTBLD), an interrupt enabling routine (INTEST), and an interrupt event notice list structure. The interrupt event notice list structure is shown in Figure 123. Only one event notice is required because the prioritization and ordering are handled automatically by INTBLD. This logic and the use of the list structure are represented by the functional flow chart shown in Figure 124. Any instruction simulation that modifies the processor status word must call INTEST to test if any interrupts have been delayed because mask bits have been disarmed. INTEST searches the delayed queue checking for previous occurrence of newly enabled interrupts; it also searches the pending queue for any pending interrupt that has been missed previously. In either case, if an interrupt is found, it is rescheduled by the appropriate call to INTBLD.

When the interrupt event occurs, it tests if the interrupt is enabled. If not enabled, the event is placed on the delayed queue. If enabled, the trap Exchange Status and PC instruction are executed, using the address contained in the event notice. In either case, the next pending interrupt is removed from the pending queue and the next interrupt event notice is placed on the future event list.

The interrupt simulation scheme just described has no constraint on either the number of interrupt levels or the prioritization scheme. In other words, the interrupt scheme is completely general and not constrained to the DP M point design.

### *f. ILS Outputs*

The outputs of the ILS consist of one print line indicating the instruction that was executed, the current state of the processor, and the instruction execution time including any

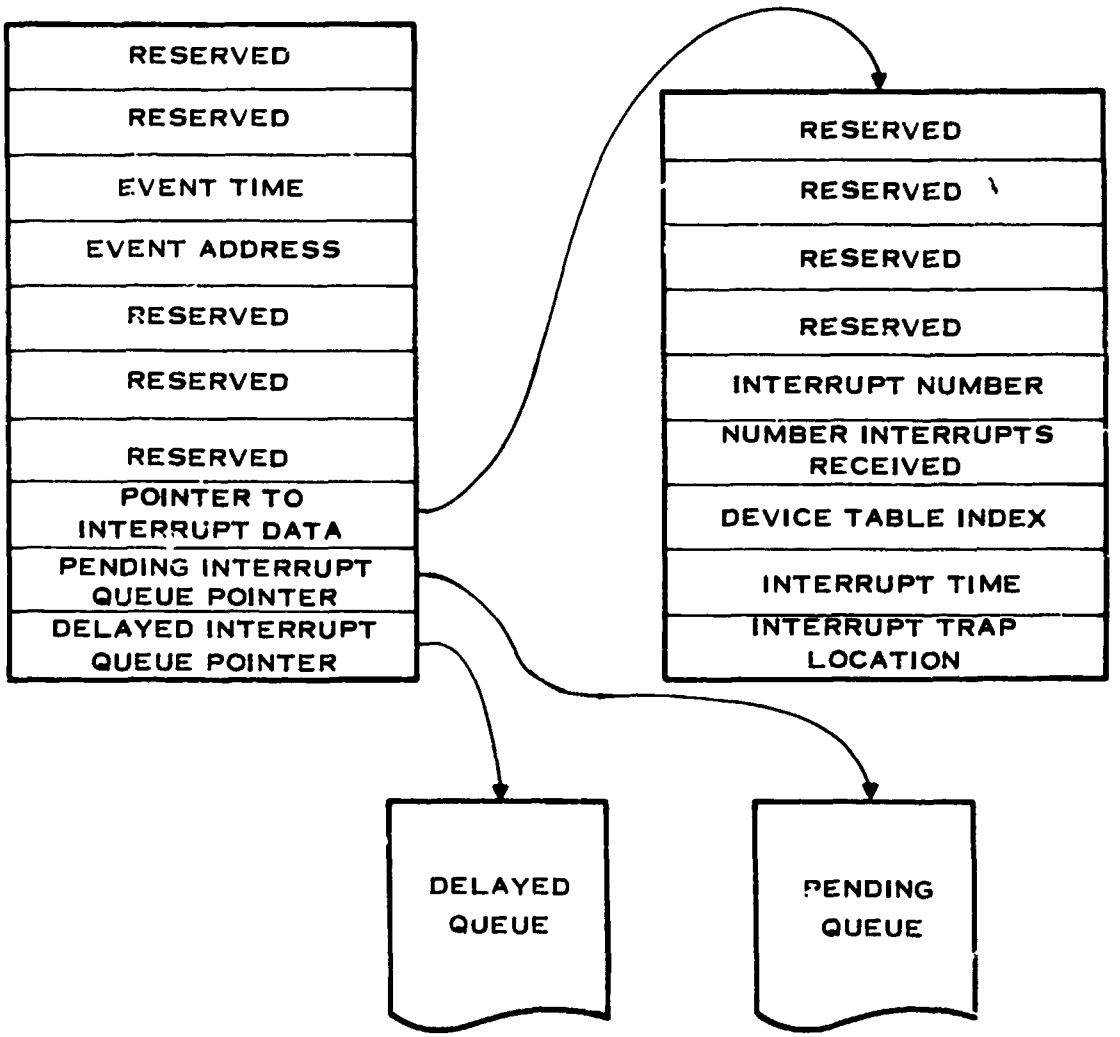



Figure 123. Interrupt Event Notice List Structure

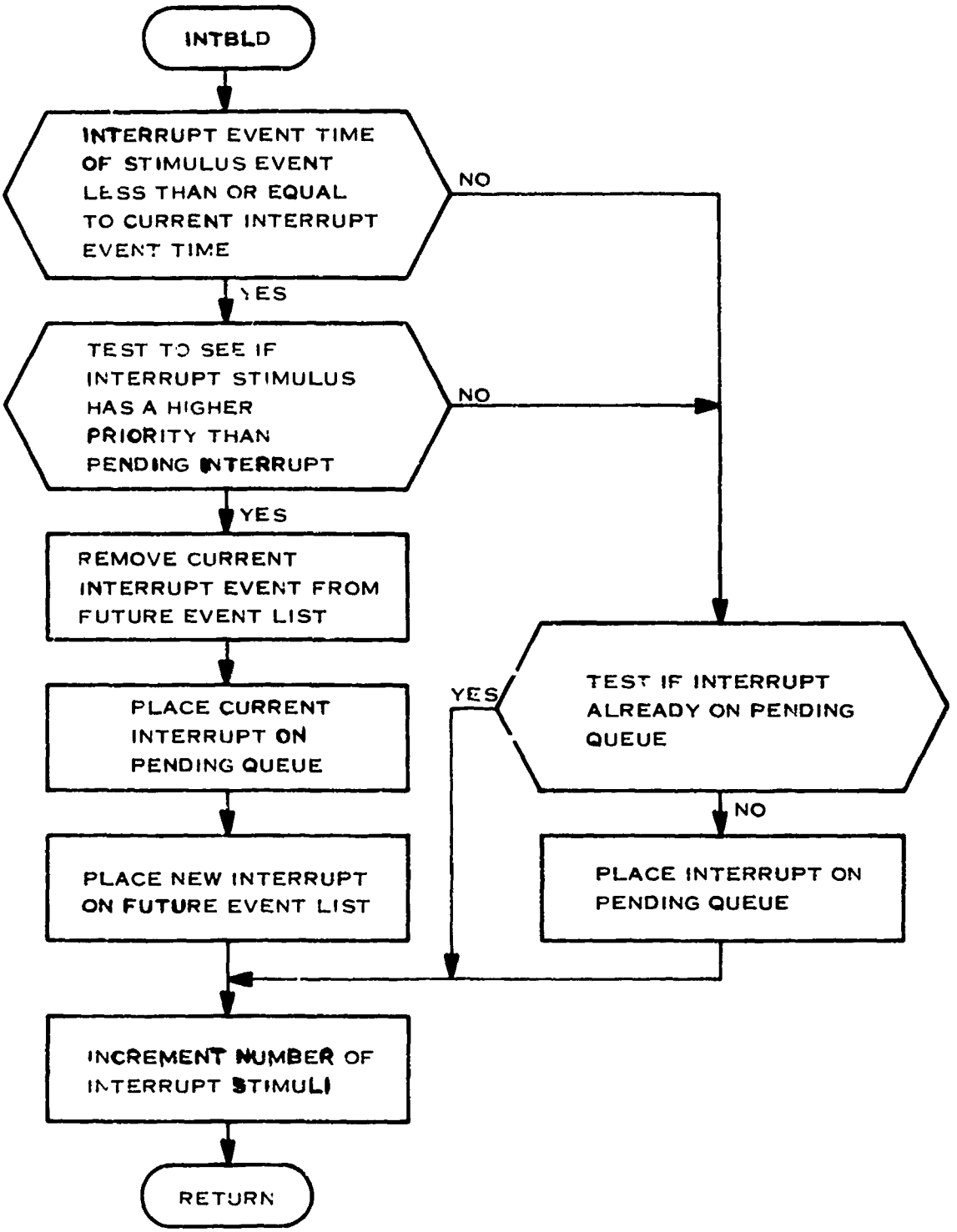



Figure 124 Interrupt Stimulus Generation Functional Flow Chart

memory conflict delays. The print output may be optionally turned off by user commands. Each portion of the output shown in Figure 125 is explained by the following short annotation:

| Column Label | Comments |
|---|---|
| PC | Value of Program Counter before instruction execution |
| MNEM | Instruction mnemonic |
| OPFLD | Operand field of instruction |
| D01 | Value of derived operand or address |
| D02 | Value of second derived operand or address (present only on double length instructions) |
| R0-R7 | Contents of processor register file. A period indicates that value did not change from previous instruction |
| SW | Contents of status word |
| MT | Amount of time memory used by memory refresh during current instruction |
| GT | Amount of time memory used by Global bus transfers during current instruction |
| LT | Amount of time memory used by Local bus transfers during current instruction |
| AT | Amount of time memory used by autonomous DMA channel during current instruction |
| TCST | Cumulative total of MT+GT+LT+AT |
| IT | Instruction execution time excluding any delay caused by memory conflicts |
| TOT TIME | Cumulative total of simulation time (Note: all times in microseconds, all other data in hexadecimal) |

Also provided are the data and time that the simulation run took place.

An optional output of the ILS is a histogram which provides a summary of all instructions executed during the simulation. The summary gives, for each instruction, the number of executions, total execution time, percentage by time, and percentage by occurrence. Also provided is a summary indicating the number as well as percentage of 16-bit and 32-bit instructions executed. Figure 126 is an example of the histogram output.

### 3. Bus Interface Unit Simulation

The Bus Interface Unit (BIU) simulation portion of the PE Simulator consists of those event routines that simulate the performance of the BIU. The BIU model is modularly structured as the integration of a set of BIU-event models, each of which is a FORTRAN subroutine representation of a discrete functional operation. Each event is modeled in terms of the effect of the event on the functional registers of the BIU hardware. The models are driven by exogenous events and also interact with other PE functional element models (e.g., the PE memory) in the performance of particular BIU-related activities, as shown in Figure 127.

INSTRUCTION LEVEL SIMULATION 14:33:29 05NOV74 PAGE 2

| PC | MNEM | OPFLD | OD1 | OD2 | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | SW | RT | GT | LT | AT | TCST | IT | TOT TIME |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 7FCF | FF81 | 007F | 0101 | 0101 | 007E | FF81 | 013D | | | | | | | | |
| 013D | CCS | 1,FF81 | FF81 | | . | . | . | . | . | . | . | 013E | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 123.0 |
| 013E | BCS | 5,013E | 013E | | . | . | . | . | . | . | . | 013F | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 125.0 |
| 013F | STR.1A | 5,3 | 0101 | | . | . | . | 0102 | . | . | . | 0140 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 128.0 |
| 0140 | L.DX | 1,FFFF,3 | 007E | | . | 007E | . | . | . | . | . | 0143 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 132.0 |
| 0142 | CCS | 1,007E | 007E | | . | . | . | . | . | . | . | 0143 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 134.0 |
| 0143 | BCS | 5,0143 | 0143 | | . | . | . | . | . | . | . | 0144 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 136.0 |
| 0144 | ST.DX | 6,FFFF,3 | 0101 | | . | . | . | . | . | . | . | 0146 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 140.0 |
| 0146 | LR.1A | 1,4 | FF81 | | . | FF81 | . | . | 0102 | . | . | 0147 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 143.0 |
| 0147 | CCS | 1,FF81 | FF81 | | . | . | . | . | . | . | . | 0148 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 145.0 |
| 0148 | BCS | 5,0148 | 0148 | | . | . | . | . | . | . | . | 0149 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 147.0 |
| 0149 | CR | 3,4 | 0102 | | . | . | . | . | . | . | . | 014A | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 149.0 |
| 014A | BCS | 5,014A | 014A | | . | . | . | . | . | . | . | 014B | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 151.0 |
| 014B | STDS | 5,FFFF,3 | 0101 | | . | . | . | . | . | . | . | 014C | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 155.0 |
| 014C | LDS | 1,FFFF,4 | 007E | | . | 007E | . | . | . | . | . | 014D | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 159.0 |
| 014D | CCS | 1,007E | 007E | | . | . | . | . | . | . | . | 014F | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 161.0 |
| 014E | BCS | 5,014E | 014E | | . | . | . | . | . | . | . | 014F | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 163.0 |
| 014F | CCS | 2,007F | 007F | | . | . | . | . | . | . | . | 0150 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 165.0 |
| 0150 | BCS | 5,0150 | 0150 | | . | . | . | . | . | . | . | 0151 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 167.0 |
| 0151 | LC | 1,8000 | 8000 | | . | 8000 | . | . | . | . | . | 0153 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 170.0 |
| 0153 | BCS | 6,0153 | 0153 | | . | . | . | . | . | . | . | 0154 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 172.0 |
| 0154 | ACS | 7,0012 | 0012 | | . | . | . | . | . | . | . | 0167 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 175.0 |
| 0167 | LCS | 0,0010 | 0010 | | 0010 | . | . | . | . | . | . | 0168 | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 177.0 |
| 0168 | LCS | 1,000F | 000F | | . | 000F | . | . | . | . | . | 0169 | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 179.0 |
| 0169 | LCS | 2,0010 | 0010 | | . | . | 0010 | . | . | . | . | 016A | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 181.0 |
| 016A | LCS | 3,0011 | 0011 | | . | . | . | 0011 | . | . | . | 016B | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 183.0 |
| 016B | CR | 0,1 | 000F | | . | . | . | . | . | . | . | 016C | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 185.0 |
| 016C | BCS | 4,016C | 016C | | . | . | . | . | . | . | . | 016D | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 187.0 |
| 016D | CR | 0,2 | 0010 | | . | . | . | . | . | . | . | 016E | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 189.0 |
| 016E | BCS | 5,016E | 016E | | . | . | . | . | . | . | . | 016F | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 191.0 |
| 016F | CR | 0,3 | 0011 | | . | . | . | . | . | . | . | 0170 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 193.0 |
| 0170 | BCS | 6,0170 | 0170 | | . | . | . | . | . | . | . | 0171 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 195.0 |
| 0171 | LC | 4,F00F | F00F | | . | . | . | . | F00F | . | . | 0173 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 198.0 |
| 0173 | CC | 4,F00E | F00E | | . | . | . | . | . | . | . | 0175 | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 201.0 |
| 0175 | BCS | 4,0175 | 0175 | | . | . | . | . | . | . | . | 0176 | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 203.0 |
| 0176 | CC | 4,F00F | F00F | | . | . | . | . | . | . | . | 0178 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 206.0 |
| 0178 | BCS | 5,0178 | 0178 | | . | . | . | . | . | . | . | 0179 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 208.0 |
| 0179 | CC | 4,F010 | F010 | | . | . | . | . | . | . | . | 0178 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 211.0 |
| 0178 | BCS | 6,0178 | 0178 | | . | . | . | . | . | . | . | 017C | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 213.0 |
| 017C | LCS | 5,0000 | 0000 | | . | . | . | . | . | 0000 | . | 017D | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 215.0 |
| 017D | C.D | 5,0155 | FFFF | | . | . | . | . | . | . | . | 017F | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 219.0 |
| 017F | BCS | 4,017F | 017F | | . | . | . | . | . | . | . | 0180 | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 221.0 |
| 0180 | LC | 6,0157 | 0157 | | . | . | . | . | . | . | 0157 | 0182 | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 224.0 |
| 0182 | LC | 5,FFFF | FFFF | | . | . | . | . | . | FFFF | . | 0184 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 227.0 |
| 0184 | CR.1 | 5,6 | 00FF | | . | . | . | . | . | . | . | 0185 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 230.0 |
| 0185 | BCS | 6,0185 | 0185 | | . | . | . | . | . | . | . | 0186 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 232.0 |
| 0186 | LCS | 0,0000 | 0000 | | 0000 | . | . | . | . | . | . | 0187 | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 234.0 |
| 0187 | LOCC | 0,5678 | 5678 | | A987 | . | . | . | . | . | . | 0189 | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 237.0 |
| 0189 | BCS | 6,0189 | 0189 | | . | . | . | . | . | . | . | 018A | 8000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 239.0 |
| 018A | CC | 0,A987 | A987 | | . | . | . | . | . | . | . | 018C | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 242.0 |
| 018C | BCS | 5,018C | 018C | | . | . | . | . | . | . | . | 018D | 4000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 244.0 |
| 018D | NC | 0,79F0 | 79F0 | | 2980 | . | . | . | . | . | . | 018F | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 247.0 |
| 018F | BCS | 4,018F | 018F | | . | . | . | . | . | . | . | 0190 | 0000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 249.0 |

Figure 125. Example of Instruction-Level Simulator Output

HISTOGRAM FOR INSTRUCTION LEVEL SIMULATION

| INSTRUCTION | ST.CR | SFTCC | SFSWC | RCC | XSPCC |
|---|---|---|---|---|---|
| NUMBER OF EXECUTIONS | 1 | 11 | 5 | 1 | 2 |
| TOTAL EXECUTION TIME | 4.0 | 139.0 | 20.0 | 3.0 | 18.0 |
| PERCENTAGE BY TIME | 0.2 | 6.7 | 1.0 | 0.1 | 0.9 |
| PERCENTAGE BY OCCURENCE | 0.2 | 1.8 | 0.8 | 0.2 | 0.3 |

* * * EXTENDED SHORT FORMAT INSTRUCTIONS * * *

| INSTRUCTION | LCS | ACS | CCS | BCS | BILR |
|---|---|---|---|---|---|
| NUMBER OF EXECUTIONS | 35 | 36 | 42 | 208 | 1 |
| TOTAL EXECUTION TIME | 70.0 | 72.0 | 84.0 | 424.0 | 4.0 |
| PERCENTAGE BY TIME | 3.4 | 3.5 | 4.1 | 20.4 | 0.2 |
| PERCENTAGE BY OCCURENCE | 5.8 | 6.0 | 7.0 | 34.7 | 0.2 |

| INSTRUCTION | IBNS |
|---|---|
| NUMBER OF EXECUTIONS | 10 |
| TOTAL EXECUTION TIME | 39.0 |
| PERCENTAGE BY TIME | 1.9 |
| PERCENTAGE BY OCCURENCE | 1.7 |

| INSTRUCTION | LOS | STOS |
|---|---|---|
| NUMBER OF EXECUTIONS | 3 | 1 |
| TOTAL EXECUTION TIME | 12.0 | 4.0 |
| PERCENTAGE BY TIME | 0.6 | 0.2 |
| PERCENTAGE BY OCCURENCE | 0.5 | 0.2 |

* * * INSTRUCTION LENGTH USAGE STATISTICS * * *

| | | | |
|---|---|---|---|
| TOTAL NUMBER OF INSTRUCTIONS EXECUTED = | 599 | | |
| TOTAL NUMBER OF LONG (32-BIT) INSTRUCTIONS EXECUTED = | 218 | 36.4 % BY OCCURRENCE | 53.1 % BY TIME |
| TOTAL NUMBER OF SHORT (16-BIT) INSTRUCTIONS EXECUTED = | 381 | 63.6 % BY OCCURRENCE | 46.9 % BY TIME |
| NUMBER OF EXTENDED SHORT INSTRUCTIONS EXECUTED = | 336 | 56.1 % BY OCCURRENCE | 34.2 % BY TIME |
| NUMBER OF 16-BIT STANDARD INSTRUCTIONS EXECUTED = | 45 | 7.5 % BY OCCURRENCE | 12.7 % BY TIME |

**Figure 126. Simulation Instruction Usage Histogram (Sheet 1 of 2)**

Basically, the BIU model is segregated into two primary functional activities: Message Input and Message Output. Each major activity is then subdivided into a collection of register level events associated with the performance of the activity during PF operation. Each event model is essentially duplicated for both Global and Local (G/L) bus activities, with minor deviations occurring in a few event models where G/L operational characteristics differ.

### *a. BIU Simulation Input Data*

Input message data for the BIU simulator consists of two card input streams arranged in chronological order that define the input message that is to be simulated for a particular simulation experiment. These data sets may be generated by hand or may be an output of an earlier System Network Simulation experiment.

FORTRAN NAMELIST input was selected to allow the data to be self-documenting as well as meaningful. Table 29 gives the format for the input message data sets. Figure 128 shows an example of input message definition data for one local and two global messages.

### *b BIU Simulation List Structures*

In keeping with the overall DPM simulation system philosophy, a series of predefined, identical event notice list structures were defined to be used by all BIU simulation events. Though not all portions of the list structures were used by all event routines, the uniformity of the structures greatly simplified program development as well as checkout

HISTOGRAM FOR INSTRUCTION LEVEL SIMULATION

| | | | |
|---|---|---|---|
| TOTAL TIME SPENT DURING SIMULATION | = | 2074.0 | |
| TOTAL TIME SPENT DURING INSTRUCTION EXECUTION | = | 2074.0 | 100.00 % OF TOTAL |
| TOTAL TIME SPENT DURING MEMORY REFRESH | = | 0.0 | 0.0 % OF TOTAL |
| TOTAL TIME SPENT DURING GLOBAL BUS TRANSFERS | = | 0.0 | 0.0 % OF TOTAL |
| TOTAL TIME SPENT DURING LOCAL BUS TRANSFERS | = | 0.0 | 0.0 % OF TOTAL |
| TOTAL TIME SPENT DURING DMA TRANSFERS | = | 0.0 | 0.0 % OF TOTAL |
| NUMBER OF INSTRUCTIONS EXECUTED | = | 549 | |
| AVERAGE INSTRUCTION EXECUTION TIME | = | 3.46 | |

* * * STANDARD FORMAT INSTRUCTIONS * * *

| INSTRUCTION | AR | AC | A.D | BC | CR |
|---|---|---|---|---|---|
| NUMBER OF EXECUTIONS | 1 | 4 | 2 | 13 | 4 |
| TOTAL EXECUTION TIME | 2.0 | 12.0 | 8.0 | 39.0 | 8.0 |
| PERCENTAGE BY TIME | 0.1 | 0.6 | 0.4 | 1.9 | 0.4 |
| PERCENTAGE BY OCCURENCE | 0.2 | 0.7 | 0.3 | 2.2 | 0.7 |

| INSTRUCTION | CC | CR.I | C.. | DVW | DVC |
|---|---|---|---|---|---|
| NUMBER OF EXECUTIONS | 61 | 1 | 1 | 8 | 3 |
| TOTAL EXECUTION TIME | 183.0 | 3.0 | 4.0 | 160.0 | 63.0 |
| PERCENTAGE BY TIME | 8.8 | 0.1 | 0.2 | 7.7 | 3.0 |
| PERCENTAGE BY OCCURENCE | 10.2 | 0.2 | 0.2 | 1.3 | 0.5 |

| INSTRUCTION | LR | LC | LR.I | LR.IA | L.D |
|---|---|---|---|---|---|
| NUMBER OF EXECUTIONS | 4 | 65 | 5 | 2 | 3 |
| TOTAL EXECUTION TIME | 8.0 | 195.0 | 15.0 | 6.0 | 12.0 |
| PERCENTAGE BY TIME | 0.4 | 9.4 | 0.7 | 0.3 | 0.6 |
| PERCENTAGE BY OCCURENCE | 0.7 | 10.9 | 0.8 | 0.3 | 0.5 |

| INSTRUCTION | L.DX | LOGR | LOGC | MR | MC |
|---|---|---|---|---|---|
| NUMBER OF EXECUTIONS | 4 | 13 | 1 | 1 | 9 |
| TOTAL EXECUTION TIME | 16.0 | 26.0 | 3.0 | 20.0 | 189.0 |
| PERCENTAGE BY TIME | 0.8 | 1.3 | 0.1 | 1.0 | 9.1 |
| PERCENTAGE BY OCCURENCE | 0.7 | 2.2 | 0.2 | 0.2 | 1.5 |

| INSTRUCTION | NR | NC | OC | SR | SC |
|---|---|---|---|---|---|
| NUMBER OF EXECUTIONS | 1 | 3 | 1 | 2 | 8 |
| TOTAL EXECUTION TIME | 2.0 | 9.0 | 3.0 | 4.0 | 24.0 |
| PERCENTAGE BY TIME | 0.1 | 0.4 | 0.1 | 0.2 | 1.2 |
| PERCENTAGE BY OCCURENCE | 0.2 | 0.5 | 0.2 | 0.3 | 1.3 |

| INSTRUCTION | S.D | SFTSC | STR.I | STR.IA | ST.D |
|---|---|---|---|---|---|
| NUMBER OF EXECUTIONS | 2 | 14 | 2 | 1 | 4 |
| TOTAL EXECUTION TIME | 8.0 | 116.0 | 6.0 | 3.0 | 16.0 |
| PERCENTAGE BY TIME | 0.4 | 5.6 | 0.3 | 0.1 | 0.8 |
| PERCENTAGE BY OCCURENCE | 0.3 | 2.3 | 0.3 | 0.2 | 0.7 |

**Figure 126. Simulation Instruction Usage Histogram (Sheet 2 of 2)**

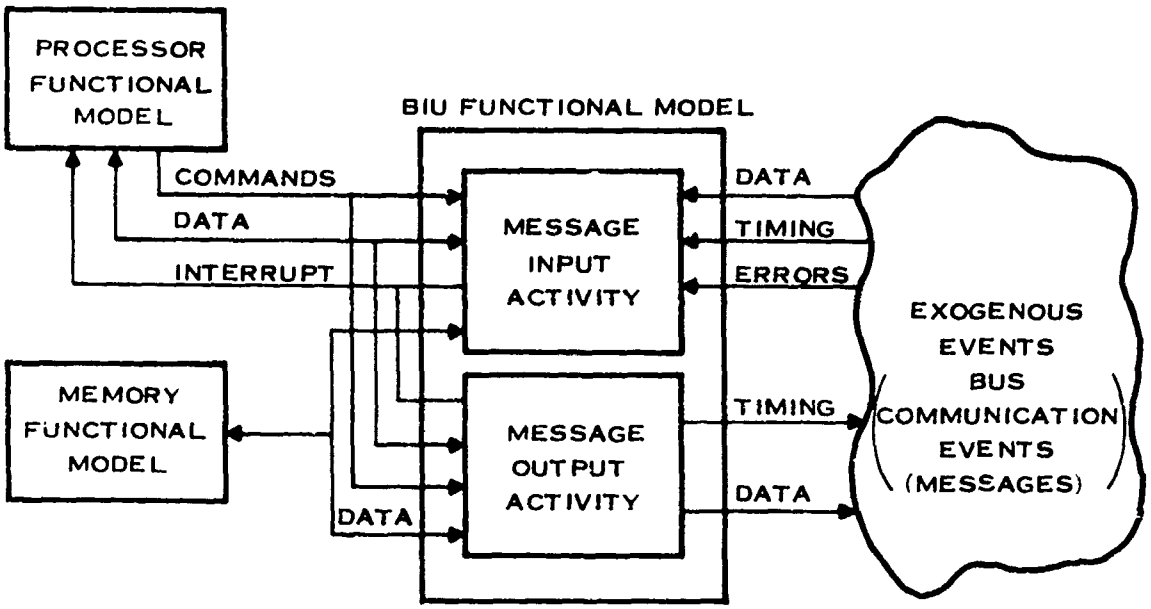



Figure 127. BIU Functional Model/PE Functional Simulation Interface

```
&LMSG
LMTIME=800,
LMLGTH=9,
LMDATA=31,0,11,0,12,0,13,0,14,0,15,0,16,0,17,0,18,0,
LLAST=.TRUE.
&END

&GMSG
GMTIME=40,
GMLGTH=5,
GMDATA=9,0,1,0,2,0,3,0,4,0,
&END
&GMSG
GMTIME=800,
GMLGTH=9,
GMDATA=31,0,11,0,12,0,13,0,14,0,15,0,16,0,17,0,18,0,
GLAST=.TRUE.
&END
```

Figure 128 BIU Input Message Data Example

**TABLE 29. BIU SIMULATION INPUT DATA SPECIFICATION FORMAT**

Col 1

*b &LMSG

Repeat for each input message

b LMTIME = t, (message start time)
b LMLGTH = n, (message length, including header)
b LMDATA = h $e_h$ $d_1$ $e_1$ $d_2$ $e_2$ ... $d_{n-1}$ $e_{n-1}$,
b LQERR = TRUE (specifies bus quiescence error)
b LLAST = TRUE, (specifies last message)
b &END

where h = header
$e_h$ = error indicator for header
$d_i$ = data error indicator = $2^{16+i}$, where
i = 0 incorrect bit length
= 1 bad parity
= 2 invalid encoding format
$e_i$ = 0 for no error

Global Bus Input Data

b &GMSG
b GMTIME =
b GMLGTH =
b GMDATA =
b GQERR =
b GLAST =
b &END

} Same meaning as in local

*b indicates blank required

Figure 129 gives the structure of the BIU event notices. IOSIMZ is a vector whose elements are pointers to the event notice list structures. Each event notice list structure required is defined at simulation initiation time and a pointer to it is placed in IOSIMZ. Defining the event notice list structure before simulation saves the overhead required to dynamically create and destroy event notices.

The reserved words are required by the Basic Simulation for all event notices. Priority is used to resolve memory conflicts for those events simulating memory reference. Memory conflict resolution will be discussed in the section on memory simulation. Next event address, reference time, and next event notice are used by the bus memory reference event to schedule successors to events that requested a memory transfer. The local global flag allows one event routine to simulate both the Local and Global bus based upon the status of the flag. The read write flag specifies whether an event is requesting a read from or a write to memory. Memory data contains the data to be written to memory, or the data returned from a memory read request.

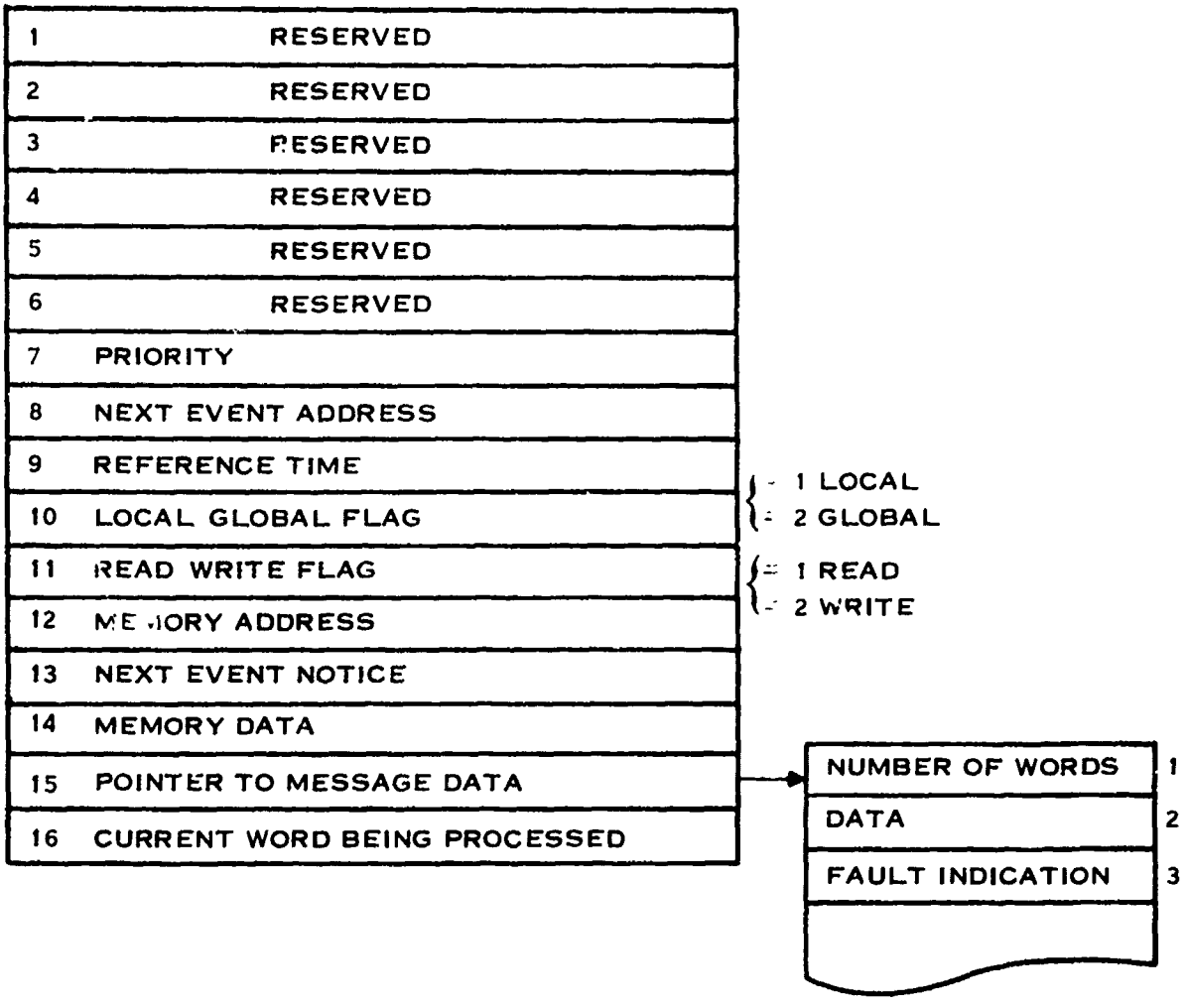



**Figure 129. BIU Simulation Event Notice List Structure**

The pointer to message data points to a dynamic memory block that contains input message data. The current word being processed indicates which word in the input data is currently being input.

As stated earlier, corresponding to each BIU event model described is one event routine that implements the logic specified by the model definition. Each model event routine schedules its successor event(s) by placing them on the future event list. Concurrency, memory conflicts, and event delays are automatically handled by the Basic Simulator and the memory simulation model.

For the sake of brevity, only one representative event routine will be exampled in detail. However, all other events are approximately equal in complexity. Figure 130 is the event model

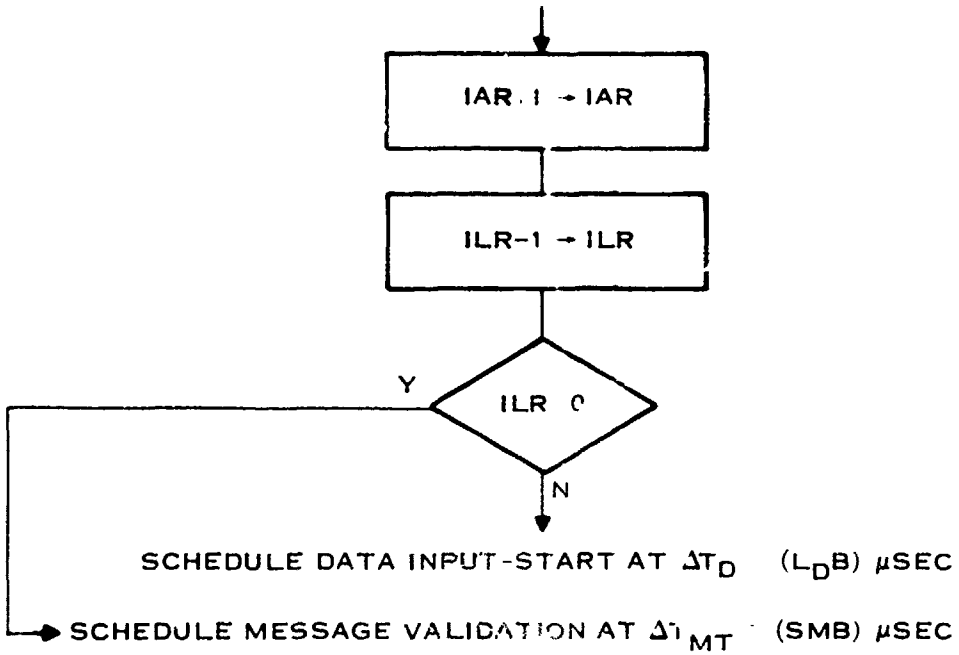



**Figure 130. Data Input End Event Model**

for the data input end event. Numbers in the following discussion refer to the FORTRAN statement numbers in Figure 131.

| | |
|---|---|
| 0001 | Event routine Bus Data Input End (BDIE) |
| 0002 | BIU simulation variables |
| 00015, 20, 23 | BIU simulation event notice list template |
| 0026 | Call routine to print status of BIU internal |
| 0027 | Fetch value of local/global flag |
| 0028 | Begin actual model logic; increment IAR |
| 0029 | Decrement ILR |
| 0030 | Test value of ILR |
| 0031 | Schedule Bus Data Input Start (BDIS) event at a time based upon the start of the last input word plus the time to transfer one data word |
| 0033 | Schedule Bus Message Validation (BMV) event at a time based upon current simulation time plus the time to transfer message sync |
| 0034 | Free dynamic memory used for input message. |

```
FORTRAN IV V LEVEL  20                  BDIE                  DATE = 79020          15/25/50

0001            SUBROUTINE BDIE                                                     BDIE0010
                C                                                                   BDIE0020
                C ------------------------------------------------------------------ BDIE0030
                C SYSTEM                                                            BDIE0040
                C     OPEN PE SIMULATION                                            BDIE0050
                C PURPOSE                                                           BDIE0060
                C     BUS DATA INPUT END EVENT ROUTINE                              BDIE0070
                C PROGRAMMER                                                        BDIE0080
                C     MILES RICHARD                                                 BDIE0090
                C DATE                                                              BDIE0100
                C     OCT 1976                                                      BDIE0110
                C DEPENDENCY                                                        BDIE0120
                C     FORTRAN                                                       BDIE0130
                C COMMONS                                                           BDIE0140
                C   BIU                                                             BDIE0150
                C     IAR     - INPUT ADDRESS REGISTER                              BDIE0160
                C     ILR     - INPUT LENGTH REGISTER                               BDIE0170
                C     B       - BIT TIME                                            BDIE0180
                C     SM      - MESSAGE SYNC TIME                                   BDIE0190
                C     LD      - DATA LOADA LENGTH                                   BDIE0200
                C     AMFLG   - ASSOCIATIVE MATCH FLAG                              BDIE0210
                C   CUREVT                                                         BDIE0220
                C     EVENTZ - CURRENT EVENT NOTICE POINTER                         BDIE0230
                C   MEMBER                                                          BDIE0240
                C     MEMORY - DYNAMIC MEMORY                                       BDIE0250
                C SUBPROGRAMS                                                       BDIE0260
                C     BDIS    - BUS DATA INPUT START EVENT                          BDIE0270
                C     RIUPRT - PRINT BIU HARDWARE REGISTER                          BDIE0280
                C     BMV     - BUS MESSAGE VALIDATION EVENT                        BDIE0290
                C     RETMEM - FREE DYNAMIC MEMORY ROUTINE                          BDIE0300
                C     SCHEDE - SEMINULE SIMULATION EVENT ROUTINE                    BDIE0310
                C LOCAL VARIABLES                                                   BDIE0320
                C     LOFLAG - LOCAL/GLOBAL FLAG  (LOCAL =1 , GLOBAL =2)            BDIE0330
                C     PTR2   - POINTER TO INPUT MESSAGE DATA                        BDIE0340
                C     PTR    - TEMPORARY POINTER VARIABLE                           BDIE0350
                C     RETTIM - REFERENCE TIME                                       BDIE0360
                C ------------------------------------------------------------------ BDIE0370
                C                                                                   BDIE0380
                C ******************************************************************* BDIE0390
                C     SUBPROGRAM STATEMENTS                                         BDIE0400
                C ******************************************************************* BDIE0410
                C                                                                   BDIE0420
0003            COMMON /BIU/  /                                                     BDIE0430
                *     IAR, ILR, IAP, ILP, TM, LM, ODP, DAP, ILD, DENFG, DAFG,     BDIE0440
                *     IPFG, BER, APR, MTCFG, AUSIU, BTYPE, READM, WRITEM, LOCAL,  BDIE0450
                *     GLOBAL, G, LM, B, SM, LD, SW, TG, W, BARM, DFFG, LSST,      BDIE0460
                *     BTYPEQ, LSSTQ, WLFF, WLFM, RDM, RDF, BQM, BWF, MTCF, MPWF,  BDIE0470
                *     MPWM, WLFF, WCFM, DPFF, DPFM, DFFF, DFFM, MRCF, MRCM, DAF,  BDIE0480
                *     DAM, DFM, IAF, MTCM, LAST, LADDD, AMFLG, MLVF, MLVM, RMEF,  BDIE0490
                *     RMEM, DUMMY                                                 BDIE0500
0004            COMMON /CUREVT/                                                     BDIE0510
                *     EVENTZ, PARMSZ                                              BDIE0520
0005            COMMON /MEMBER/                                                     BDIE0530
                *     MEMORY                                                      BDIE0540
                C                                                                   BDIE0550
                C ------------------------------------------------------------------ BDIE0560
                C                                                                   BDIE0570
0006            INTEGER                                                             BDIE0580
                *     RADR, ADP, MTM, MLOC(2), DPFC(2), BWP, BCA, BTYPE(2),       BDIE0590
                *     BTYPQ(2), AUSIU(2), DFFG(2), DFEF, DFEM, DPEF, DPEM,        BDIE0600
                *     DUMMY(6), G, GLOBAL, MPWM, MPRM, IDR(2), IPR(2), IDP(2),    BDIE0610
                *     ISR(2), LM, LOCAL, LSSDD(2), LSST(2), LADDD(2), LAFT(2),    BDIE0620
                *     MRM3DY(1053), RWBF, RMEM, MLVF, MLVM, MRCF, MRCM, MTCF,     BDIE0630
                *     MTCFG(2), MTCM, DAF, DAFG(2), DAM, DAR(2), DDR(2), DBF, DFM, BDIE0640
                *     DTBF(2), DLD(2), DPFG(2), PARMSZ, READM, SM, TC, W, WCEF,   BDIE0650
                *     WCFM, WLFF, WLFM, WRITEM                                    BDIE0660
                C                                                                   BDIE0670
                C ------------------------------------------------------------------ BDIE0680
                C                                                                   BDIE0690
0007            INTEGER             AMFLG     (2)                               BDIE0700
0008            EQUIVALENCE         B                                           BDIE071C
                C     ENTRY         BDIE                                          BDIE0720
0009            EXTERNAL            BDIS                                        BDIE0730
0010            EXTERNAL            RIUPRT                                      BDIE0740
0011            EXTERNAL            BMV                                         BDIE0750
0012            INTEGER             EVENTZ                                      BDIE0760
0013            INTEGER             IAR       (2)                               BDIE0770
0014            INTEGER             ILR       (2)                               BDIE0780
0015            INTEGER             LD                                          BDIE0790
0016            INTEGER             LOFLAG    (1)                               BDIE0800
0017            EQUIVALENCE         LOFLAG    (1), MEMORY(1))                   BDIE0810
0018            EXTERNAL            MEMORY                                      BDIE0820
0019            INTEGER             PTR2      (1)                               BDIE0830
0020            EQUIVALENCE         PTR2      (1), MEMORY(1))                   BDIE0840
0021            INTEGER             PTR                                         BDIE0850
0022            INTEGER             RETTIM    (1)                               BDIE0860
0023            EQUIVALENCE         RETTIM    (1), MEMORY(10))                  BDIE0870
0024            EXTERNAL            SCHEDE                                      BDIE0880
0025            INTEGER             SM                                          BDIE0890
                C                                                                   BDIE0900
                C ******************************************************************* BDIE0910
                C     EXECUTABLE STATEMENTS                                         BDIE0920
                C ******************************************************************* BDIE0930
                C                                                                   BDIE0940
0026            CALL BIUPRT                                                         BDIE0950
0027            PTR = LOFLAG(EVENTZ)                                                BDIE0960
0028            IAR(PTR) = IAR(PTR) + 1                                             BDIE0970
0029            ILR(PTR) = ILR(PTR) - 1                                             BDIE0980
0030            IF (ILR(PTR) .LE. 0) GO TO 101                                      BDIE0990
0031            CALL SCHEDE (EVENTZ, RETTIM(EVENTZ)+DUMM, INTERVENE(1))             BDIE1000
                C ------------------------------------------------------------------ BDIE1010
0032            RETURN                                                              BDIE1020
                C ------------------------------------------------------------------ BDIE1030
0033          101 CALL SCHEDE (EVENTZ, RETTIM(EVENTZ)+DUMM, INTERVENE(2))           BDIE1040
0034            AMFLG(PTR) = 0                                                      BDIE1050
0035            CALL MEMBER (PTR,(EVENTZ))                                          BDIE1060
                C ------------------------------------------------------------------ BDIE1070
0036            RETURN                                                              BDIE1080
                C ------------------------------------------------------------------ BDIE1090
0037            END                                                                 BDIE1100
```

**Figure 131. Data Input End Event Routine**

```
                        INSTRUCTION LEVEL SIMULATION                    09:17:12        21OCT74                 PAGE    5

      PC   MNEM      OPFLD      DD1  DD2   R0   R1   R2   R3   R4   R5   R6   R7   SW  MT  GT  LT  AT    TCST    IT  TOT TIME
                                            0000 0000 0000 0000 0000 0000 06FE 0170

   EVENT BMTS    (LOCAL)  >>>                                                                                          64.5

                        LOCAL    GLOBAL                  LOCAL    GLOBAL                  LOCAL    GLOBAL
              IDR   =     0033      001E   BLR   =        0003      000A   ODR   =        0000      0028
              IAR   =     0000      1FC1   BPR   =        0002      000A   OAR   =        0000      0179
              ILR   =     0000      0000   IFR   =        0000      0000   OLR   =        0000      0002
              IQP   =     0777      077E   ISR   =        CFE1      FFE3   OENFG =        0001      0001
             MTCFG  =     0001      0001   OPFG  =        0000      0000   OAFG  =        0000      0001
        0170 ACS      4,0001         0001          .    .    .    .  0001    .    .  0171 0000 0.0 0.0 0.0 0.0     4.0  2.0      65.0
        0171 MIC      000A,0143,4    0000          .    .    .    .    .    .    .  0173 8000 0.0 0.0 0.0 0.0     4.0  5.0      70.0

<<< EVENT BDOS    (GLOBAL) >>>                                                                                         70.5

                        LOCAL    GLOBAL                  LOCAL    GLOBAL                  LOCAL    GLOBAL
              IDR   =     402F      001E   BLR   =        0003      000A   ODR   =        0000      0028
              IAR   =     0000      1FC1   BPR   =        0002      000A   OAR   =        0000      0179
              ILR   =     0000      0000   IFR   =        0000      0000   OLR   =        0000      0002
              IQP   =     0777      077E   ISR   =        CFE1      FFE3   OENFG =        0001      0001
             MTCFG  =     0000      0001   OPFG  =        0000      0000   OAFG  =        0000      0001
+++ GLOBAL BUS MEMORY READ  DATA = 0029 FROM ADDRESS = 0179 +++                                                        70.5

<<< EVENT BDOE    (GLOBAL) >>>                                                                                         71.5

                        LOCAL    GLOBAL                  LOCAL    GLOBAL                  LOCAL    GLOBAL
              IDR   =     402F      001E   BLR   =        0003      000A   ODR   =        0000      0028
              IAR   =     0000      1FC1   BPR   =        0002      000A   OAR   =        0000      0179
              ILR   =     0000      0000   IFR   =        0000      0000   OLR   =        0000      0002
              IQP   =     0777      077E   ISR   =        CFE1      FFE3   OENFG =        0001      0001
             MTCFG  =     0000      0001   OPFG  =        0000      0000   OAFG  =        0000      0001
        0173 ACS      4,0001         0001          .    .    .    .  0002    .    .  0174 0000 0.0 1.0 0.0 0.0     5.0  2.0      73.0
        0174 STSWR.IA 0,6            8FFF          .    .    .    .    .    .  06FF 0175 8FFF 0.0 0.0 0.0 0.0     5.0  4.0      77.0
        0175 LR.IA    7,6            0116          .    .    .    .    .    .  0700 0116 8FFF 0.0 0.0 0.0 0.0     5.0  3.0      81.0
        0116 LCS      0,FFF6         FFF6     FFF6 .    .    .    .    .    .    0117 8FFF 0.0 0.0 0.0 0.0     5.0  2.0      83.0

<<< EVENT BARR    (LOCAL)  >>>                                                                                         84.5

                        LOCAL    GLOBAL                  LOCAL    GLOBAL                  LOCAL    GLOBAL
              IDR   =     402F      001E   BLR   =        0003      000A   ODR   =        0000      0029
              IAR   =     0000      1FC1   BPR   =        0002      000A   OAR   =        0000      017A
              ILR   =     0000      0000   IFR   =        0000      0000   OLR   =        0000      0001
              IQP   =     0777      077E   ISR   =        CFE       FFE3   OENFG =        0001      0001
             MTCFG  =     0000      0001   OPFG  =        000.      0000   OAFG  =        0000      0001
+++ LOCAL BUS MEMORY READ  DATA = 0001 FROM ADDRESS = 1FC2 ++                                                         85.0

<<< EVENT BAMT    (LOCAL)  >>>                                                                                         86.0

                        LOCAL    GLOBAL                  LOCAL    GLOBAL                  LOCAL    GLOBAL
              IDR   =     402F      001E   BLR   =        0003      000A   ODR   =        0000      0029
              IAR   =     1FC2      1FC1   BPR   =        0002      000A   OAR   =        0000      017A
              ILR   =     0000      0000   IFR   =        0000      0000   OLR   =        0000      0001
              IQP   =     0777      077E   ISR   =        CFE1      FFE3   OENFG =        0001      0001
             MTCFG  =     0000      0001   OPFG  =        0000      0000   OAFG  =        0000      0001
```

Figure 132. Example of Bus Interface Unit Simulation Output

**TABLE 30. CAW OPERATIONS**

| Operation (I/O Instruction Function) | Read/Write (R/W) |
|---|---|
| Output Primary Level Interrupt Mask | W |
| Input Primary Level Interrupt Mask | R |
| Output (G/L) Interface Status | W |
| Input (G/L) Interface Status | R |
| Activate (G/L) Output | W |
| Output (G/L) Bus Control | W |
| Input Present (G/L) Position | R |
| Input (G/L) Queue Pointer | R |
| Input (G/L) High-Priority Interrupts (Level 3/5) | R |
| Input (G/L) Low-Priority Interrupts (Level 4/6) | R |
| Enable (G/L) Output | W |
| Disable (G/L) Output | W |
| Internal Operations: | |
| Activate Autonomous Transfer | W |
| Deactivate Autonomous Transfer | W |
| Input ATC Status, Upper Half | R |
| Input ATC Status, Lower Half | R |
| Input Programmable Interval Timer | R |
| Output Programmable Time Interval | W |
| Input and Deactivate Interval Timer | R |
| External Operations: | |
| Programmed Data Output | W |
| Programmed Data Input | R |

NOTE: (G/L) = Global/Local

### *c. BIU Simulation Output*

The output of the BIU simulation routines is interleaved with the output of the ILS. Each occurrence of a BIU event routine causes a line to be printed, indicating which event routine has occurred and whether it was processing as a Local or Global bus as well as the time of occurrence. The status of all internal BIU registers for both the Local and Global bus are also displayed as they exist at the beginning of the event.

All memory transfers initiated by BIU event routines cause a line to be printed at the initiation of the transfer. This line indicates which bus initiated the transfer, whether the transfer was a read or a write, the data, the address, and the time the transfer was initiated.

Figure 132 is an example of the output generated by the BIU simulation.

### 4. I/O Simulation

The I/O simulation section of the PE Simulator consists of those event routines necessary to simulate the performance of the autonomous channel, processor controlled I/O, interval timer operation, and pseudo I/O operations.

The processor CAW assignments given in Table 30 provides the interface between the I/O routines and the ILS. When a processor I/O instruction is executed, the CAW and data are passed to a routine that simulates the action of that CAW. The result of the CAW simulation may invoke other event routines such as autonomous channel transfer event or acknowledge event.

Detailed modeling of the internal operation of the autonomous channel is not required to provide accurate simulation. The only operation that affects PE Performance is memory conflicts. Since memory conflict resolution is handled automatically, it is possible to implement the functional performance of the I/O system without detailed physical simulation of internal I/O register transfers.

The Programmable Interval Timer (PIT) is a program-controlled counter that generates an interrupt when its contents have been decremented to a count of zero. The PIT is initiated by an Output Programmable Time Interval CAW which inserts a program specified time interval into the counter. The current value of the PIT is read at any time by execution of an Input Programmable Interval Timer CAW. Execution of an Input and Deactivate Programmable Interval Timer CAW will terminate the PIT activity and read the current timer value. Execution of an Output Programmable Time Interval CAW while a previous time interval is in progress will cause abortion of the current countdown process and re-initialization of the PIT. The PIT has a resolution of 50 microseconds.

The simulator provides program control capability via the pseudo-input/output instruction (assembler mnemonic PIO). An unused op-code is used to represent the PIO instruction. The Simulator recognizes the op-code and performs the I/O or control specified. The type of operation to be performed is indicated by the R-field of the PIO instruction. The R-fields and their meanings are listed below:

| R-Field | Operation Performed |
|---|---|
| 0 | Restore previous trace flag |
| 1 | Print 120 alphanumeric characters |
| 2 | Generate path usage statistics |
| 3 | Read 80-column card |
| 4 | Save trace flag and start trace |
| 5 | Save trace flag and stop trace |
| 6 | Dump |
| 7 | Terminate simulation |

*PRINT (PIO 1,A)* The contents of 60 consecutive pseudo-memory locations (120 alphanumeric characters) are printed on the next available line of the output listing when the simulator processes a PIO 1 instruction. The address field of the PIO instruction is the address of the first word to be printed.

*USAGE (PIO 2,A)* The "generate path usage statistics" pseudo-command allows statistics to be generated concerning dynamic program flow. Sixteen unique flow points are allowed and are indicated by the address field (A) of the instruction by the values 0 to 15. When the PIO 2,A-instruction is executed, the flow point specified by the A-field is incremented by one. At the completion of simulation, if the histogram (H) option was specified, a report is written indicating the number of times each flow point was executed.

*READ (PIO 3,A)* The character content of an 80-column card is stored into 40 consecutive pseudo-memory locations when the simulator processes a PIO 3 instruction. The address field of the PIO-instruction is the address where the first word is to be stored. The locations will receive packed ASCII characters.

*TRACE (PIO 4,A, PIO 5,A, PIO 0,A)* The trace is the primary output of the Simulator. The trace consists of a printed listing of the machine conditions which exist following the simulation of each instruction. The trace output is produced after the execution of each instruction when the current value of

the trace flag (TRCFLG) is one. No trace output is produced if the current value of TRCFLG is zero.

The value of TRCFLG is initially set by the Monitor program. The value of TRCFLG may be changed during simulation by use of the PIO instruction. PIO 4 sets the value of TRCFLG to one. PIO 5 sets the value of TRCFLG to zero.

It is also possible to restore a previous value to TRCFLG using PIO 0. When a PIO 4 or PIO 5 is executed, the present value of TRCFLG is saved at the location in an array (TRFBUF) defined by the three least significant bits of the address field of the PIO instruction before the new value is placed in TRCFLG. The PIO 0 instruction restores TRCFLG from TRFBUF, using the three least significant bits of the address field as an index. This allows the programmer to either trace or not trace subroutines and then restore the trace flag to its previous value in the calling routine. Subroutines may be nested up to eight levels deep while still maintaining individual control over tracing in each subroutine.

If a PIO 0 is executed before a PIO 4 or PIO 5 for a given level, the result is indeterminate.

*DUMP (PIO 6,A)* The DUMP is the secondary output of the simulator. The DUMP consists of a printed listing of the contents of 16 or more consecutive pseudo-memory locations. A DUMP may be requested in the following two ways:

- Through the Monitor
- By use of the PIO 6 instruction.

The DUMP requested through the Monitor is provided only at the termination of the simulation program. This DUMP is a listing of the contents of all of the pseudo-memory locations. This DUMP is accomplished by monitoring the DUMP flag (DMPFLG) after the completion of simulation. If DMPFLG = 1, the DUMP is provided; if DMPFLG = 0, no DUMP is provided.

The DUMP requested via the PIO 6 instruction allows the user to specify a pointer to a set of dump limits. The address field of the PIO 6 instruction is used as the pointer to a set of two consecutive memory locations. The first memory location contains the start address and the second location contains the stop address of that section of pseudo-memory to be dumped.

If a PIO 6 instruction references a set of DUMP limits which are undefined, at of range, or in an abnormal sequence, all of pseudo-memory is dumped.

Regardless of the start and stop limits given, the DUMP consists of a listing given on 16 word boundaries in increments of 16 words.

*TERMINATE SIMULATION (PIO 7,0)* When a PIO 7 is encountered by the simulator, a normal termination message is printed along with the address of the terminating instruction. The dump flag is checked, and DUMP is printed if requested before control is returned to the Monitor.

## 5. Memory Simulation

The one central, shared resource in the PE is the memory. At any one time, there may be several entities requesting a memory transfer. Consequently, the memory simulation must provide for conflict resolution and subsequent delaying of memory access to those requesters of lower priority. Since *a priori* prediction of memory conflicts is not possible, some mechanism must exist for resolving conflict within the different memory request routines themselves. The value of the priority element indicates the relative ordering of priority of memory references. It is the responsibility of any event routine that simulates a memory reference to first verify that no other event with a higher priority is currently pending. If there is a higher priority event, the lower priority event must reschedule itself and relinquish control to the higher priority event. When a memory reference event finally gains control, it must then call the delay routine (DELAY) to move any pending memory requests on the FEL between current simulation time and the length of the memory transfer to the end of the memory transfer. This delaying of memory request events is because a memory transfer request can only be honored on memory cycle boundaries. The automatic delaying also provides a mechanism for studying the effect of different length memory transfer times for reads and writes. The amount of time to delay other memory request events, i.e., the cycle time, is an argument of the DELAY routine.

The PE Simulator uses a pseudo-memory of n X 16-bit words to represent the PE memory space. The value of n is currently 2048 but may be modified to allow any memory size up to maximum addressing capability of the PE (65,536). Whenever an instruction references memory, the address actually referenced is modulo the pseudo-memory size. All memory addresses are calculated by means of a FORTRAN routine (ADDR) that computes the actual pseudo-memory address (as an index) modulo the current pseudo-memory size. This ensures a valid address. Currently, no error indication is provided to indicate an address "out-of-range." However, any level of address checking may be provided by simply modifying the ADDR routine. Isolation of address computation also allows for future modeling of various memory mixes (ROM, RAM, RMM, etc.) whose access time may vary.

## 6. Ancillary Routines

Several ancillary routines were developed to provide for ease of control and definition of experiments for the Processing Element Simulator. These routines include a monitor, assembler, and loader/link editor. Figure 133 shows the interface between the ancillary routines and the simulator.

The monitor controls the job flow to provide user flexibility and to permit batch processing of jobs. The type of options that may be invoked by the job cards include:

Assemble a source deck

Load one or more object decks

Simulate

Produce a post-simulation memory dump

Punch a relocatable object deck

Produce an instruction usage and execution flow histogram

Define I/O environment.

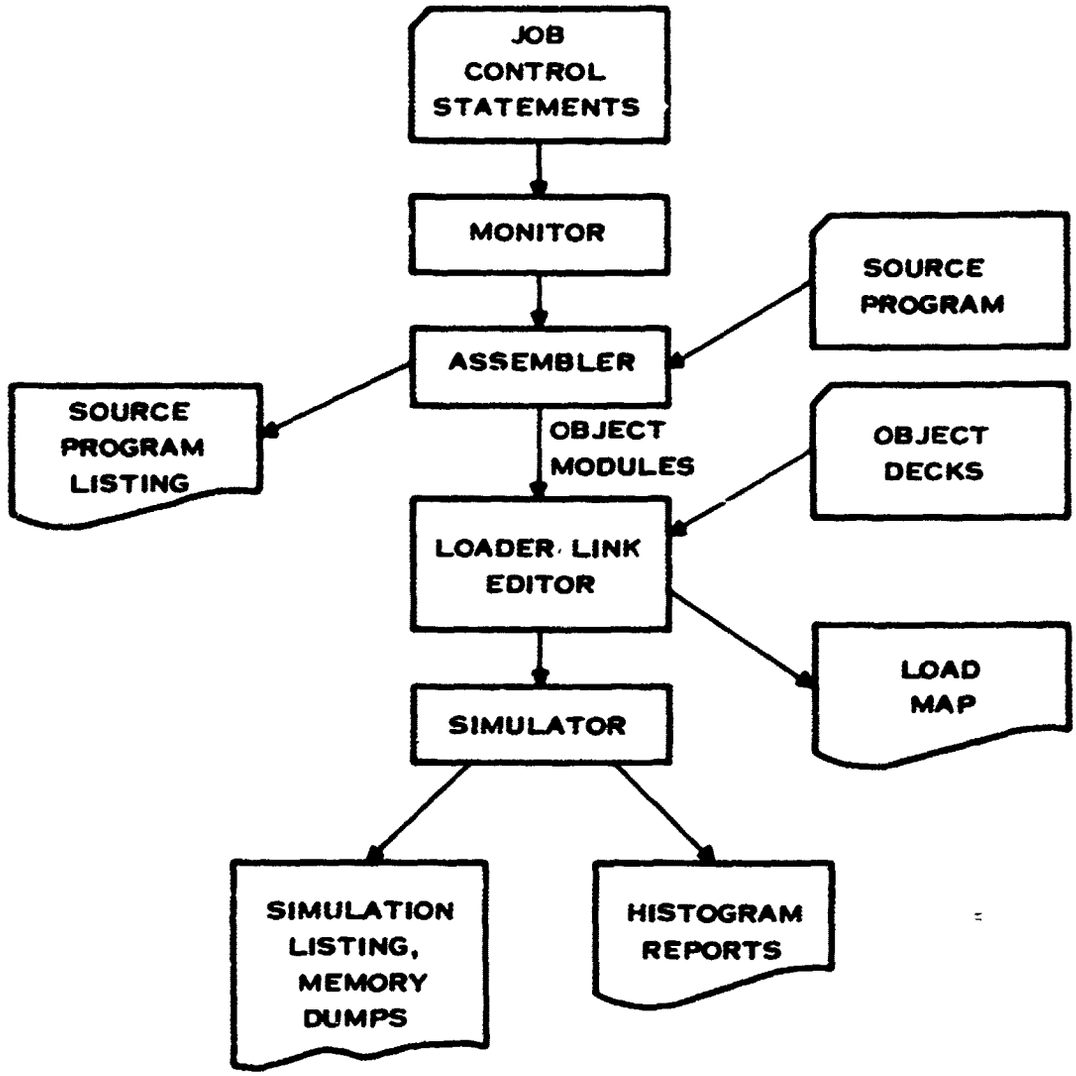



**Figure 133. System Control Flow Diagram**

The assembler accepts source instructions from cards and generates assembly listings and object modules. The assembler features include:

Assemble relocatable object programs

Indicate errors and issue appropriate error messages

Produce assembly listings of the source code, object code, and program size, memory storage requirements, and instruction and data symbols with a cross-reference between names and each reference to their use

External symbol references

Assembler directives

```
    INSTRUCTION LEVEL DIAGNOSTIC - V 1.0                    TIME: 14:31:42    05NOV74     PAGE     6

LOC   OBJECT CODE     CARD IMAGE                                                                CARDNUM

                      PAGE                                                                  V1 01350    135
                     .*************************************************************************V1 C136C    136
                     .                                                                      V1 01370    137
                     .   5.0  STANDARD DERIVED OPERAND (LOAD) AND ADDRESS (STORE) TESTS     V1 C1380    138
                     .              ASSUMES:          R0,R2,R4,R6=-127   R1,R3,R5=+126      V1 C1390    139
                     .              MODIFIED   ON EXIT: R0= R5 & (R1=-127;R2=+127;         V1 01400    140
                     .                                  R3,R4=TSTMEM+1   CCR=C             V1 C141C    141
                     .                                                                     V1 C1420    142
                     .*************************************************************************V1 01430    143
                     .                                                                      V1 C1440    144
                     .        TEST FOR PROPER STEPPING OF PC DURING A LOAD IMMEDIATE        V1 C1450    145
                     .                                                                      V1 C146C    146
0129  0188           .        LR.IA R0,PC              LOAD R0 IMMEDIATE VALUE BY USING     V1 C147C    147
                     .                                      LOAD REG(PC) INDIRECT AUTOINCREV1 01480    148
012A  7FCF           .        BS    $                  PC = PC+1  ERROR TRAP                V1 C149C    149
                     .                                                                     V1 C14AC    150
                     .        NOTE THAT IF PC <  PC+1  ERROR THEN THE PROGRAM WILL LOOP     V1 01510    151
                     .              BACK TO SOME PREVIOUS ADDRESS. SIMILARLY IF IT BRANCHES V1 0152C    152
                     .              TOO FAR FORWARD THE NEXT INSTRUCTION WILL BE SKIPPED    V1 C153C    153
                     .              WHICH WILL BE TESTED FOR LATER                          V1 C154C    154
                     .                                                                     V1 C155C    155
012B  5FC2           .        LCS   R2,+127             PC+2  < PC <  PC+N  TEST SETUP      V1 0156C    156
                     .                                                                     V1 C157C    157
                     .        USE LOAD IMMEDIATE AND LOAD REGISTER TO SET UP MEMORY ADDRESS V1 01580    158
                     .        POINTERS. INITIALIZE AND TEST MEMORY WITH REGISTER INDIRECT   V1 0159C    159
                     .        LOAD AND STORE.                                               V1 C160C    160
                     .                                                                     V1 C161C    161
012C  0188 0131      .        LC    R3,TSTMEM           LOAD IMMEDIATE  MEMORY POINTER      V1 C162C    162
012E  011C           .        LR    R4,R3               LOAD REGISTER  MEMORY POINTER       V1 0163C    163
012F  025E           .        STR.I R6,R3               STORE REGISTER INDIRECT  INITIALIZE V1 01640    164
0130  0161           .        LR.I  R1,R4               LOAD REGISTER INDIRECT  READ        V1 0165C    165
0131  6051           .        CCS   R1,-127             TEST FOR CORRECT VALUE              V1 01660    166
0132  7FDD           .        BCS   NE,$                    AND STOP IF NOT                 V1 0167C    167
                     .                                                                     V1 01680    168
                     .        TEST LOAD AND STORE, DIRECT AND REGISTER INDIRECT, AND CCR    V1 01690    169
                     .                                                                     V1 C170C    170
0133  02C5 0131      .        ST.D  R5,TSTMEM           STORE DIRECT                        V1 01710    171
0135  7FDD           .        BCS   NE,$              CHECK THAT STORE DOES NOT CHANGE CCR  V1 0172C    172
0136  0159           .        LR.I  R1,R3               LOAD REGISTER INDIRECT              V1 0173C    173
0137  7FDC           .        BCS   AP,$              CHECK THAT LOAD DOES CHANGE CCR       V1 01740    174
0138  5F91           .        CCS   R1,+126           TEST FOR CORRECT VALUE                V1 0175C    175
0139  7FDD           .        BCS   NE,$                    AND STOP IF NOT                 V1 0176C    176
013A  0266           .        STR.I R6,R6               STORE REGISTER INDIRECT             V1 01770    177
013B  01C1 0101      .        L.D   R1,TSTMEM           LOAD DIRECT                         V1 C178C    178
013D  6051           .        CCS   R1,-127             TEST FOR CORRECT VALUE              V1 01790    179
013E  7FDD           .        BCS   NE,$                    AND STOP IF NOT                 V1 C180C    180
                     .                                                                     V1 C181C    181
                     .        TEST LOAD AND STORE DIRECT INDEXED AND REG INDIRECT AUTOINC   V1 01820    182
                     .                                                                     V1 C183C    183
013F  029D           .        STR.IA R5,R3              STORE REGISTER INDIRECT AUTOINCREMEV1 C1840    184
0140  0109 FFFF      .        L.DX  R1,-1,R3            LOAD DIRECT INDEXED                 V1 C185C    185
0142  5F81           .        CCS   R1,+126             TEST FOR CORRECT VALUE              V1 C186C    186
0143  7FDD           .        BCS   NE,$                    AND STOP IF NOT                 V1 0187C    187
```

**Figure 134. Assembly Listing Example**

```
* * *  MEMORY USAGE SUMMARY  * * *

                        DECIMAL        HEX
INSTRUCTION OCCURRENCES:
      16-BIT              364         016C      63.4 % OF TOTAL
      32-BIT              210         00D2      36.6 % OF TOTAL
                        ------        ------
STORAGE:
INSTRUCTIONS              784         0310      72.7 % OF TOTAL
CONSTANTS                  29         001D       2.7 % OF TOTAL
RESERVED                  266         010A      24.7 % OF TOTAL
                        ------        ------
                        ------        ------
TOTAL STORAGE            1079         0437
```

**Figure 135. Static Memory Usage Report**

Location counter origin initiation, static mix of instructions, data, and reserved storage report.

Figure 134 is an example of an assembly listing. Figure 135 is an example of the static memory usage report.

The loader/link editor permits the programmer to write small modular programs and to then combine them to form the final absolute program to be loaded into PE memory. The loader/link editor optionally will also generate an absolute object deck suitable for conversion to paper tape.

## 7. Summary

As stated previously, the objective was to design, develop, and use a collection of simulation tools to assist the design and evaluation of the DP/M architecture. The programs developed proved to be adequate for the tasks to be performed. The System Network Simulator is applicable to a wide range of system configurations without major modifications because of its flexible data specification capability. Detailed reports writing capabilities exist that vary from event-by-event traces to post-simulation summaries. All report information is written so that it can be saved for later detailed analysis. The System Network Simulator was also used to augment the Process Construction study effort. Simulation experiments were used to evaluate the partitioning generated by the Process Construction algorithms. Finally, and probably most importantly, the structure of the System Network Simulator provides a mechanism for function design *and* system development by use of the concept of top down design coupled with successive, more detailed simulation.

The Processing Element Simulator is a stand-alone system that allows several levels of testing. Because of its similar structure, it can be used in conjunction with the System Network Simulator whereby information generated by the System Network Simulator is used to drive the Processing Element Simulator, or vice versa. The PE Simulator can be used to study the effect of alternative bus control schemes by modification to existing models or key parameters such as bus bit rate, word size, etc. Similarly, the effect of the various key parameters associated with the Processing Element such as memory speed, clock rate, etc., can also be investigated. Finally, the System may be used as a software development and debug tool whereby actual application software is assembled and simulated with test driver programs. The detailed reports provided by

the Processing Element Simulator such as static memory usage and dynamic instruction usage histograms allow gathering of valuable statistical information that can be used to refine the processing element instruction set, evaluate the results of different algorithms, etc.

In summary, the Functional Simulation System developed consists of two logically similar discrete-event-oriented simulators designed to simulate the particular DPM point design and also provide the flexibility to be used for other development activities

## SECTION VIII

## EXECUTIVE DESIGN AND SYSTEM OPERATION

This section summarizes the DP/M Executive design effort. The use of distributed Processing Elements (PEs) in the DP/M system concept dictates the need for a method of scheduling activities, transferring bus messages between PEs and general system control. These operations are referred to as Executive functions; the basic requirements for a DP/M executive structure were investigated and a functional design specification was developed. Considerations were given to design flexibility, expected avionics processing environments, and adaptability to different configurations of PEs distributed on the DP/M dual-level bussing network. Simulation models were developed for use in the System Network Simulator, and key modules were coded in PE assembly language to determine PE memory requirements and likely execution times for typical Executive operations. Methods of initializing a network of distributed PEs were investigated and candidate hardware and software bootstrap load procedures were defined.

### A. EXECUTIVE DESIGN CONSIDERATIONS

The role of the Executive software within the context of the DP/M system concept is to provide a set of basic control functions necessary to schedule avionic mission software routines and provide a common communication mechanism for inter-PE data transfers. The Executive must be adaptable to a variety of possible PE network configurations. This requirement for adaptability and flexibility must be viewed in light of the expected processing capabilities of the DP/M hardware. Ideally, the Executive should impose minimum computational overhead for itself on the hardware resources while providing minimum but necessary control operations to ensure satisfactory performance of the PE network in accomplishing avionic processing in a timely manner. The DP/M Executive must operate within the capabilities (and adhere to the restrictions) of the system hardware resources and the modes of likely avionic system operation within a mission. In addition, the structure or organization of real-time avionic programs dictates that the Executive provide the necessary means of scheduling, monitoring, and providing data set management between cooperating processing tasks.

The key characteristics of the DP/M system that influence the Executive design include:

- The homogenous PEs with medium computational speed, dedicated I/O to an attached external device, and typical program and data storage of 4 to 8K words per PE
- A dual redundant switchable Global bus interface per PE
- A dedicated Local bus connected to all PEs in an Affinity Group (AG)
- One or more PEs dedicated to a given processing function
- The potential use of a mass memory device for initial program load of read/write memory
- A programmable distributed bus protocol for Global and Local bus data transmission control.

The limited size of PE program and data memory, combined with the medium instruction execution times, dictates a simple and efficient Executive design. The possible physical separation of PEs in a network also requires that the Executive provide a timely method of transmitting

data between PEs. This ability to configure PEs in multiple networks imposes the requirement that the DP/M Executive structure be adaptable to various topological interconnections with minimum modifications and maximum expectation the newly created control software will function properly. Avionic processing with DP/M results in a natural partitioning of system operations along functional lines (i.e., navigation, weapon delivery). The dedication of a system external device to a PE I/O interface also encourages this functional dedication of PEs to tasks associated with a sensor and the algorithms necessary to process data from the sensor to effect a mission function. This dedicated nature of fixed functions has a substantial advantage with respect to system management. Design is simplified as is the control function when a heavy emphasis is not placed on maximum use of one resource (usually the processor hardware). The processor hardware is not the primary requirement in system concept formulation, design, and application since the projected cost, size, weight, and power advantages of the DP/M hardware offer a system designer this degree of freedom in applying processing resources to his problem.

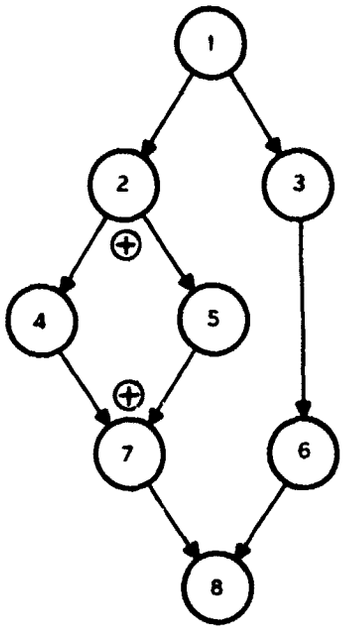
These DP/M system baseline concepts coincide with the real-time computing characteristics typically found in avionics processing systems. Within the avionics processing environment, every activity (its inputs, response, and outputs) is known and also planned for. The worst case set of conditions in terms of system loading, resource allocation, and response time is predictable. This well-ordered nature of avionics processing is in contrast to ground-based computer systems that depend upon run-time allocation of resources to satisfy user demand input requests and maximize system use. If, for example, in a ground-based system, dynamic allocation of resources is invoked by the executive control scheme, a simple activity can suffer undue penalities (its request to run is delayed) while awaiting the allocation of necessary data sets, compilers, memory, and execution time. If an error occurs and the necessary system resources cannot be made available, the requested activity may be rescheduled with little penalty to the user. However, allocation of avionic processing resources must be known, certain, and timely. The penalty for failure can endanger the success of the mission, the aircraft, and the pilot. This complete knowledge of the operating environment and capabilities of an avionic processing system places well-defined bounds on the functions necessary for DP/M Executive control, and provides a clear contrast between avionic control functions and multi-purpose ground-based operating systems.

Throughout this executive design discussion, reference will be made to terminology which arises from the use of a directed graph model. A directed graph consists of a set of nodes and a set of directed edges between these nodes. A node will be used to represent a set of computations which, once initiated, can run to completion without waiting for completion of another set of computations also represented by a node. An edge from node i to node j means that, upon completion of the computations represented by node i, the computations by node j can be initiated. The implementation of a conditional branch in this model is accomplished by an exclusive-OR symbol (⊕). Use of the symbol in conjunction with node i implies one of the following conditions:

Upon completion of the computations associated with node i, only one of the set of successor nodes can be initiated.

Completion of only one of the nodes associated with an incident edge is necessary for initiation of node i.

These concepts are illustrated in the example of Figure 136. In this example, completion of node 1 indicates that nodes 2 and 3 can be initiated. Upon completion of node 2, either node 4 or node 5 can be initiated, and the completion of only one of these is sufficient to initiate




EXAMPLE OF A DIRECTED GRAPH

**Figure 136. Example of a Directed Graph**

node 7. Node 8, on the other hand, requires the completion of both node 6 and node 7 for its initiation. Stated differently, nodes 2 and 3 are successors of node 1, and nodes 6 and 7 are predecessors of node 8.

With respect to avionic mission processing, three levels of directed graphs can be used. At the highest level, a node represents a mission- or pilot-oriented function such as navigation. For any given function, a set of nodes or options may exist to effect the function. These nodes are referred to as subfunctions (e.g., navigation modes can include inertial navigation, Loran, or doppler navigation). The intent of the subfunction definition is to coincide with the normal division and classification of avionic computer programs. At the lowest level, a subfunction is represented by a set of related tasks. In the DP/M Executive structure, a task represents an executable application software module. An example of a task within the Loran subfunction is the computation associated with converting time difference data to latitude/longitude coordinates. These concepts are illustrated in Figure 137.

The representation of software execution sequences via a directed graph concept has a particular advantage within the DP/M concept. The subfunction directed graph reveals potential process construction options in allocating tasks and program to PEs. If any one task or set of tasks must be partitioned among several PEs, the options available in allocating this software to PEs are clearly defined within the graph. Data sets that are passed from one task to another represent Local bus messages if their respective tasks are not collocated in the same PE. Likewise, collocated tasks need not generate bus traffic with their data interchanges.

The subfunction directed graph (Figure 137) contains the necessary information from which the Executive can determine task-scheduling conditions (based upon required predecessor events) and inter-task communication. Two types of directed graphs can be used to describe avionic functions: the sequential graphs and the parallel graph. A parallel graph reveals where within execution segments of code parallelism can be used. A sequential program graph removes parallelism and shows transition paths from one start node to an end node with multiple nodes emanating from a single node having an associated probability of transition. Section IX, Process Construction, discusses in more detail these types of directed graphs. Data derived from other types of graphs can be used to derive Executive design parameters; however, the Affinity Group (AG) concept is most effectively used when parallel graphs are used for scheduling purposes. Another desirable feature from the Executive design viewpoint is to allow specification of the size of a task (i.e., number of instructions, words of memory, execution time) to be flexible. Ideally, tasks will be identified in a way to encourage increased system performance. It is not the intent of the DP/M Executive design to require a certain (minimum) number of tasks to be

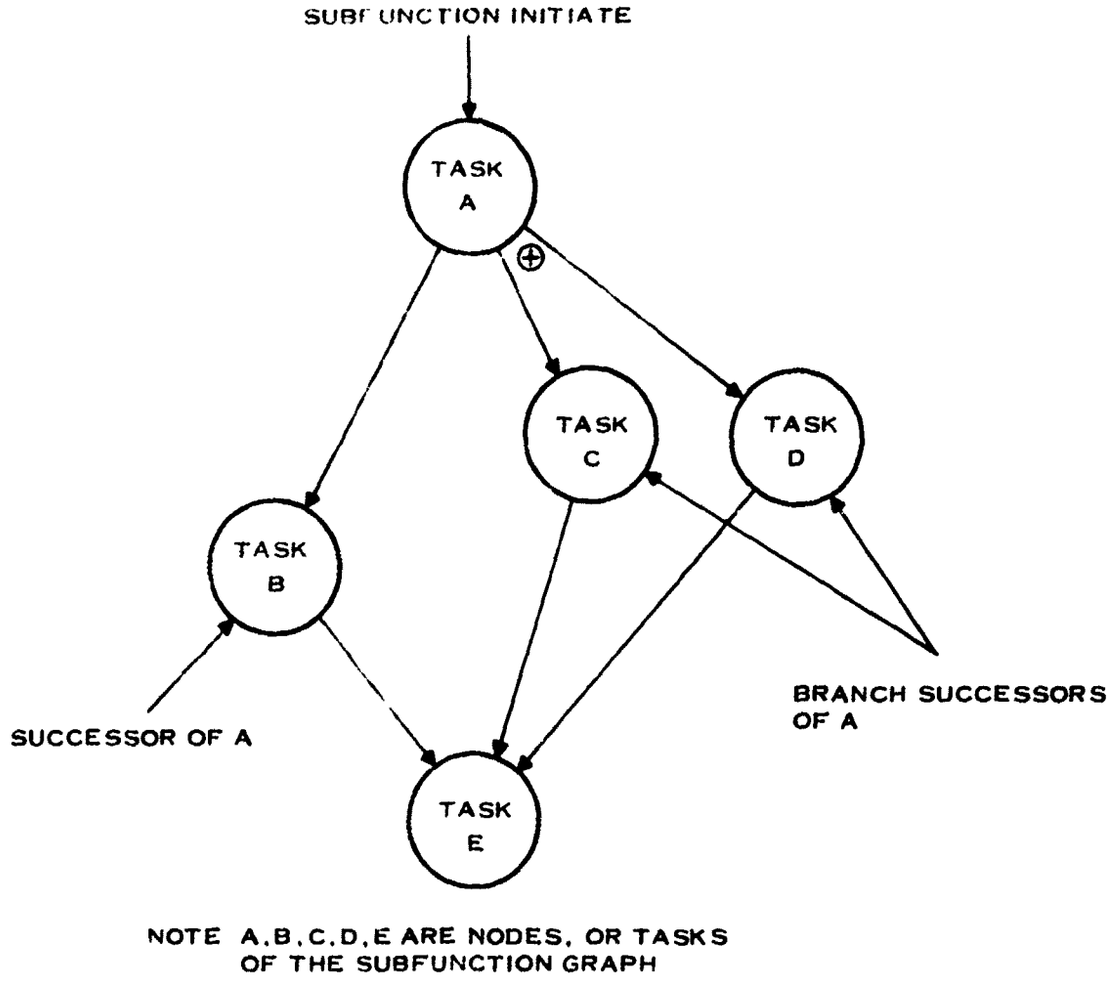



**Figure 137. Directed Graph Representation of a Subfunction**

created for a subfunction. If a subfunction can be most efficiently controlled by the Executive as a single task, this option is available. If a subfunction has a potential (or requirement) for parallelism, the Executive has a method of facilitating control of multiple PEs executing code for the different tasks of the same subfunction. It follows that, as avionic subfunctions are assigned to PEs, one PE will contain the subfunction start-node. This task would receive any "subfunction initiate" commands, and start the execution cycle for the given avionic software algorithms.

## B. EXECUTIVE CONTROL OVERVIEW

From these thoughts, two modes of operation and control were considered for the DP/M baseline system operation: a tightly coupled system operation and a loosely coupled system control philosophy. The tightly coupled philosophy assumes system order is to be directed from a central point in terms of scheduling and synchronization of functions. One function, the

Global Executive (GEX) is given the responsibility of scheduling, synchronizing, and ensuring that the system will have a predictable performance and be capable of allocating necessary resources to meet the worst case demands on system performance. Several other benefits result from a tightly coupled operating philosophy. The system can be diagnosed both on the bench in system checkout and in in-flight testing. By imposing a predictable sense of order, the system can be easily manageable without undue overhead in terms of hardware and software.

The loosely coupled system control philosophy assumes that each subfunction in the system is self-scheduling and controlling, thereby minimizing the requirement for a central control program. Consequently, each function broadcasts its data upon completion, and receiving functions must be ready for the data when it comes. In the event of random or unplanned events, synchronization between subfunctions can become difficult. Such a system also has multiple states and conditions that could conceivably occur and cause perturbations in system operation. Predicting worst case event occurrences can be difficult, as well as predicting known states in the system for checkout and testing purposes.

Each of these operating philosophies had their merits and influenced the design of the DP/M Executive. A common feature of both control schemes is the use of "directed graph-like" predecessor/successor conditions for the scheduling of tasks. The tightly-coupled philosophy assumes a common system coordination point that uses timing relationships as a necessary condition for process initiation. The loosely coupled philosophy does not depend on time, but only on the completion of program run time events (e.g., data-set-generated, prior tasks have completed, etc., hereafter referred to as predecessor conditions). The DP/M Executive design provides both capabilities. The need for a central control point for the scheduling and synchronization of functions is provided by the Global Executive (GEX). The scheduling of tasks within a PE solely on the completion of prior predecessor conditions (and not on run time boundaries) is the responsibility of the Local Executive (LEX).

The software structure capable of satisfying this dual-level control structure criterion of flexibility, reliability, transferability, and ease of testing, is a table-driven Executive. A table-driven Executive provides a separation of system logic and application software modules, so modifications can be isolated to individual modules, and table data can be readily modified when required. The complete separation of executive and applications functions provides better reliability and ease of testing and maintainability. This scheme also facilitates the development of common modules for the executive function in all PEs, since the uniqueness of the Executive is in its tables. This also eliminates the requirement for a special Affinity Group Executive which would be resident in a PE in an Affinity Group. In the baseline DP/M system, the executive tables in each PE contain all the information pertinent to the tasks assigned to that PE: for example, inputs to the task, predecessors, successors, outputs, double buffering requirements, etc.

In summary, a Global Executive initializes the system, monitors system performance, and coordinates inter-functional relationships. It schedules on a time and predecessor basis. The Local Executive schedules tasks within a PE (subfunction) based on predecessor relationships, directs the task input and output flow of information, verifies task completions, and schedules Performance Assurance Tests when a PE has completed all avionics processing for that cycle. One local Executive per subfunction must return a subfunction complete message to the Global Executive. Further details of executive design are discussed in subsequent paragraphs. The GEX, LEX scheduling philosophy is shown in Figure 138. Note that, in addition to time, predecessors such as "pilot initiate," "function complete," or messages from other functions must be satisfied for the GEX to schedule a function.

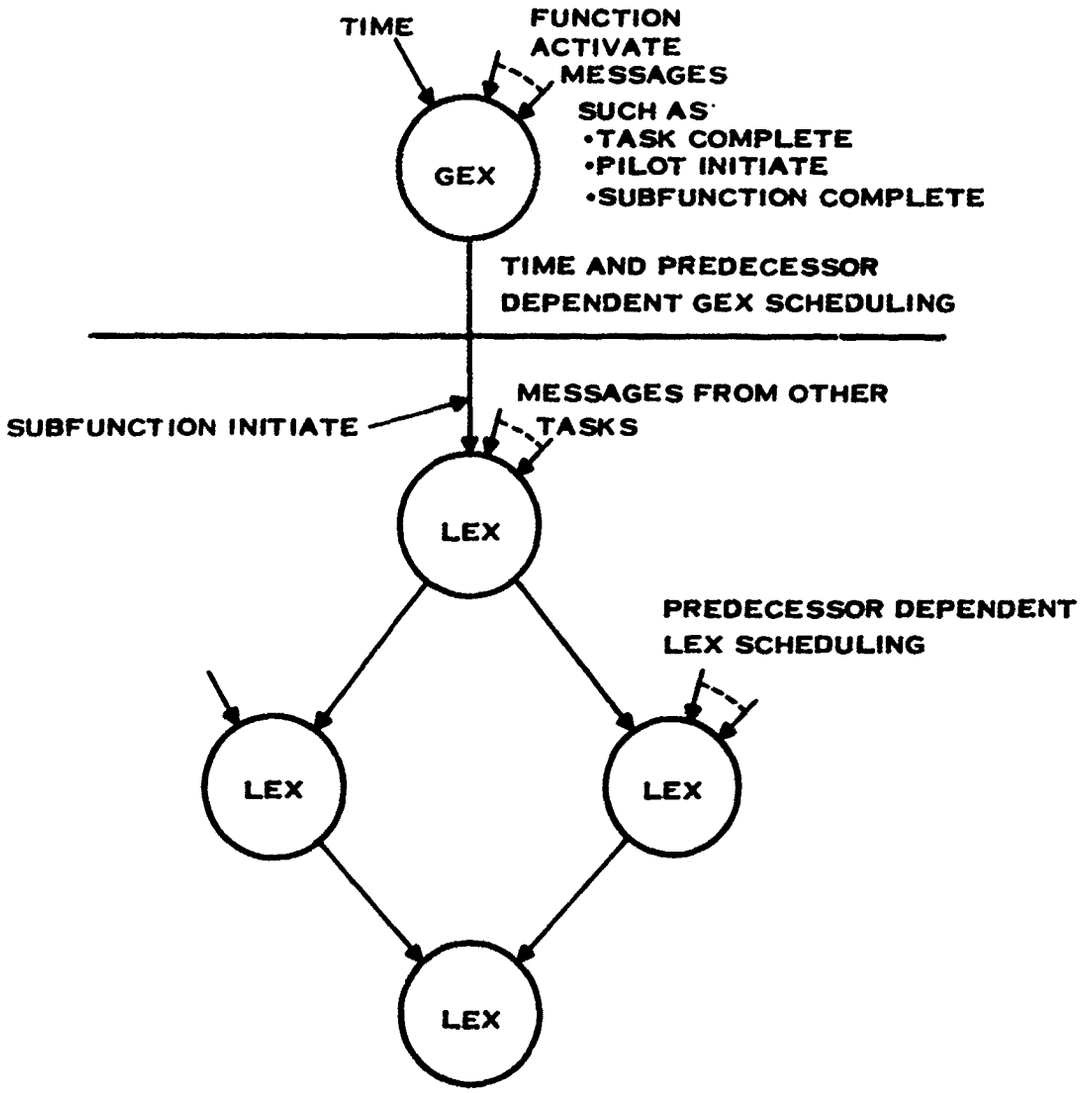



Figure 138. GEX LEX Scheduling Philosophy

## C. EXECUTIVE STRUCTURE HIERARCHY

The DP/M Executive structure provides two levels of control: the Global Executive (GEX) and the Local Executive (LEX). Functionally, the GEX assumes the role of system monitor and scheduler. It enforces subfunction interrelationships and is responsible for system performances by coordinating those software programs required to effect mission avionic functions for the pilot and aircraft. The LEX is a PE-oriented function responsible for sequencing and controlling tasks assigned to a PE. The LEX is concerned with scheduling those tasks assigned to its PE, based upon successful satisfaction of all the tasks given predecessor conditions. The Executive control hierarchy for DP/M is shown in Figure 139. This figure does not represent the individual routines within the Executive; rather, it shows the inter-relationships between major functions in the executive control hierarchy.

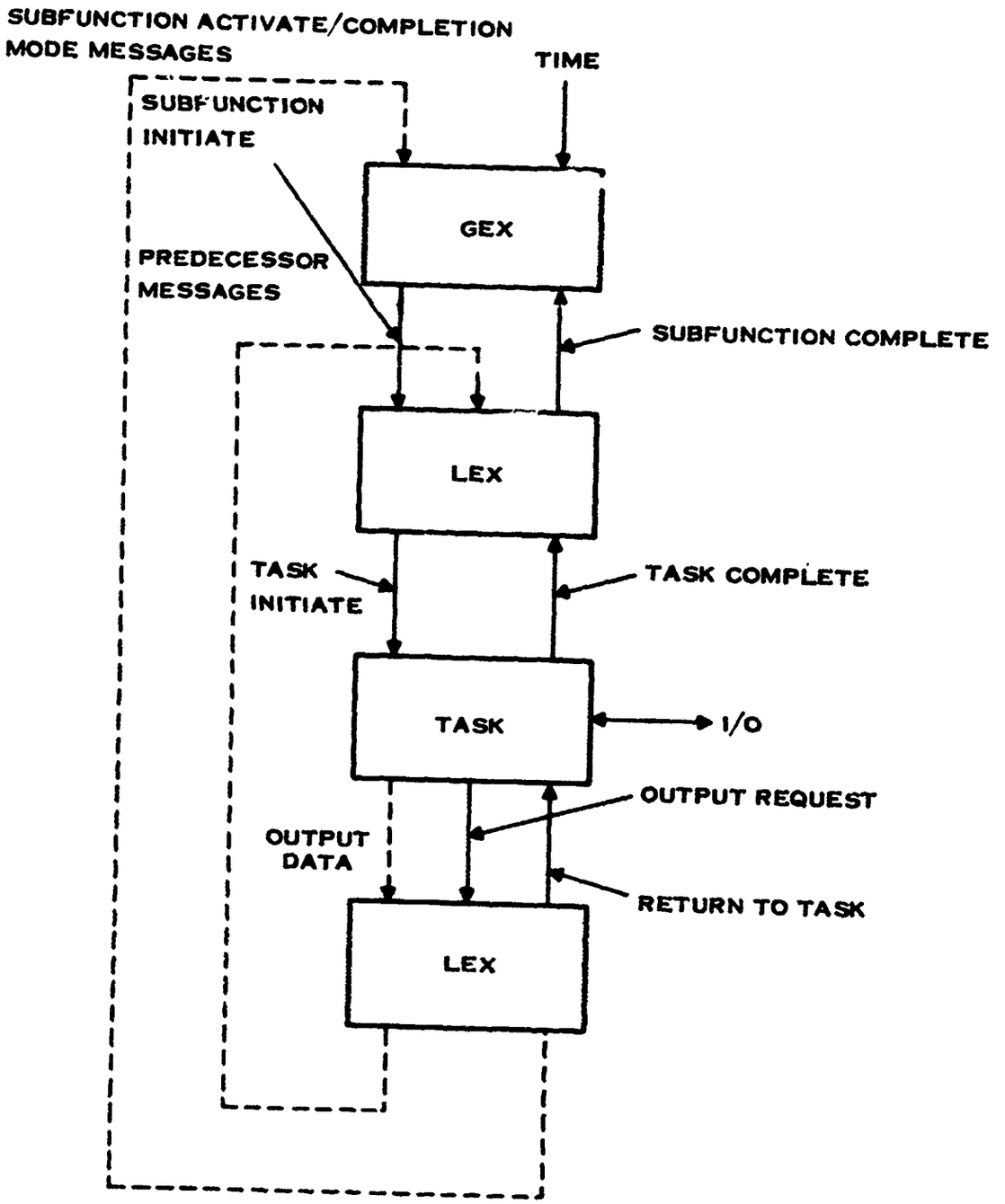



Figure 139. Executive Control Hierarchy

The Global Executive initiates a subfunction when all the subfunction predecessor requirements are satisfied and when it is time for it to run. These predecessors, in addition to start time, would be activation messages from one or more other subfunctions. The "initiate" message is transmitted by the GEX to the Local Executive on the Global bus. The Local Executive in the PE which contains the start node of the subfunction recognizes this message and, if all predessors to that start mode are satisfied, it initiates the task. The task will execute, and should it generate a data set, it will call the LEX for the disposition of the data set. The LEX takes the necessary action to route the message over the Local or Global bus or returns control to the task immediately should the message be for a task co-resident in the same PE. The LEX message handler always releases control back to the task. When a task completes, it returns control to the Local Executive, and the cycle repeats. Once a subfunction has been time initiated by the Global Executive, it will run to completion under the control of the Local Executive(s). Should an I/O device be connected to a PE, it is the responsibility of the task which uses this I/O device to control this I/O device. This does not preclude the existence of a common I/O handler routine which is used by multiple tasks.

## D. FUNCTIONAL EXECUTIVE DESIGN OVERVIEW

The following discussion provides an overview of the function provided by the DP/M Executive. Subsequent paragraphs describe each Executive module in greater detail and provide a brief summary of the design considerations used in specifying module responsibilities. Detailed flow charts for each routine are found in the Functional Specification for the DP/M Executive that is supplied as engineering data.

### 1. LEX Functional Design

The DP/M system has a homogenous set of LEX modules in each PE. The relationship among the modules of the LEX is shown in Figure 140. Each LEX initiates tasks, honors output message requests from tasks, and routes messages and data for the tasks. The LEX is table-driven, thereby maintaining the modular nature of the DP/M system and providing separation of system logic and application software modules. A Local Executive data base (or the LEX tables) in each PE contains all information pertaining to the correct initialization and control of the tasks within that PE.

Major routines found in the LEX include:

- The Bus Interrupt Service Routines which service interrupts produced by incoming messages on the Local and/or Global bus.
- The Task Scheduler composed of five sub-modules which service different aspects of the scheduler function. The Task Scheduler is responsible for determining which tasks have their predecessor requirements satisfied and are, hence, ready to be given control.
- The Dispatcher module which transfers control to the highest priority task in the dispatcher queue.
- The Output Message Interpretor module which is called by an applications task when it needs output message service. This module returns control to the task after servicing the request.

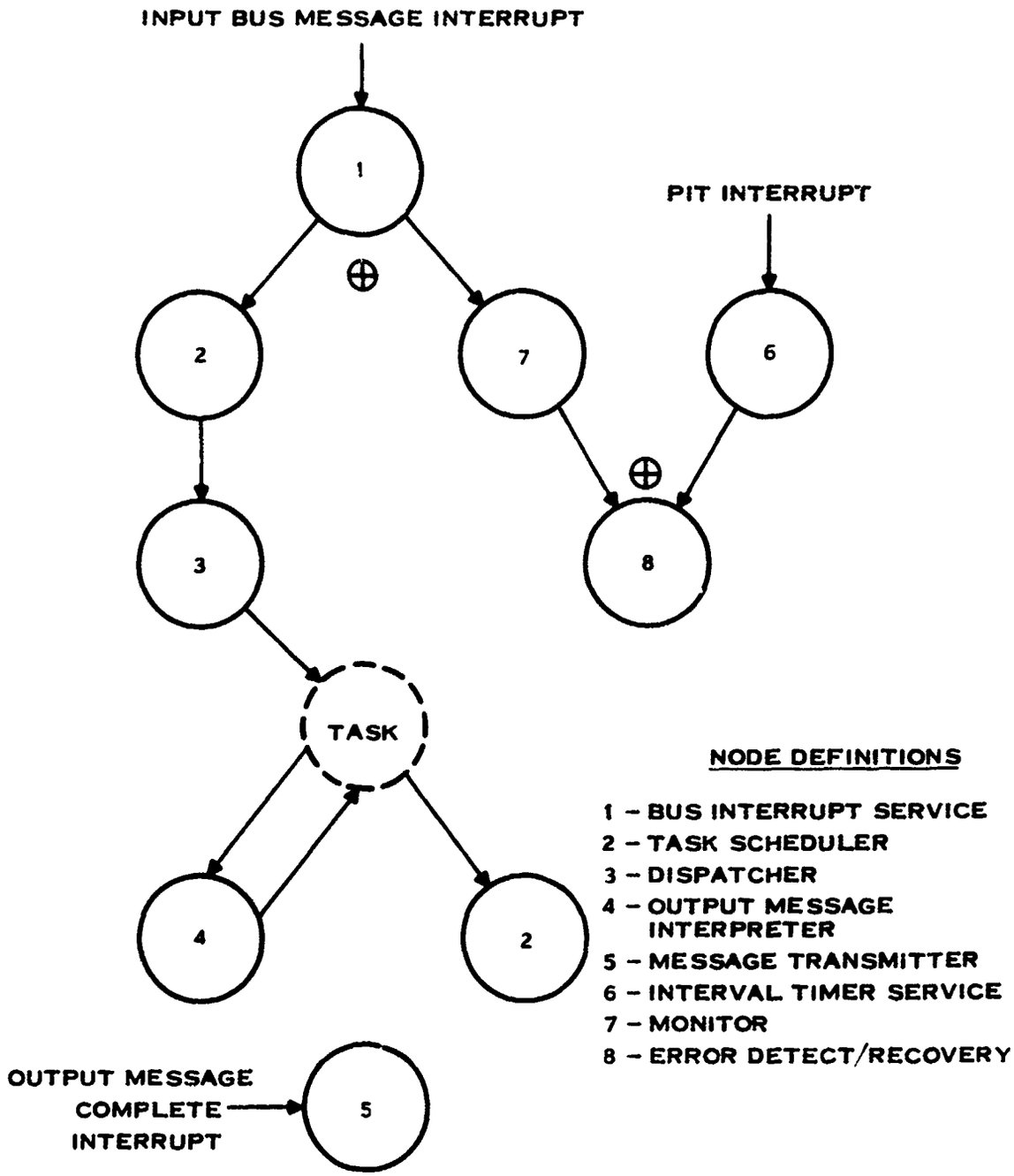



Figure 140. LEX Module and Task Interrelationship

The Message Transmitter Module which is invoked by the Output Message Complete Interrupt from one of the bus interfaces. The module is able to autonomously output data over the appropriate Local or Global bus should such a request be posted in its service queue.

The Interval Timer Service Module which is invoked when a hardware Programable Internal Timer (PIT) interrupt occurs. This is an error condition indicating that a task has taken too long to complete. The module returns an appropriate status to the error monitor module for disposition.

### 2. GEX Functional Design

The Global Executive schedules time-dependent subfunctions in the DP/M system. A time-ordered linked list provides the GEX with the relative times to schedule every time-dependent subfunction in the system. A "go" message is generated by the GEX to a subfunction only if other predecessor conditions for the subfunction have been satisfied when it is time to run the subfunction. The GEX data base (or tables) contain all information pertaining to the initialization and control of all time-dependent subfunctions in the DP/M system. Figure 141 is a diagram of routines used by the Global Executive.

The GEX Scheduler is invoked by an interrupt from the PIT. The scheduling algorithm processes a time-ordered linked list of subfunctions that are candidates for scheduling by the GEX scheduler.

All the LEX modules described previously form a part of the GEX. Certain GEX modules such as the Completion Status Monitor and the Activate/Deactivate subfunction monitor can be scheduled by the LEX in the Global Executive PE in the same manner as if they were application tasks.

The Completion Status Monitor module of the GEX is scheduled to run when a completion message is received from a subfunction. This module supplies this information to the GEX scheduler.

The Activate/Deactivate (subfunction) Monitor of the GEX is scheduled when an activate/deactivate subfunction message is received over the Global bus. Like the Completion Status Monitor, the activate/deactivate monitor supplies this information to the GEX scheduler.

The Mode Change Detector module is scheduled when a mode change command is received over the Global bus. A mode change command provides a quick method of initiating a new set of subfunctions such as might be involved when the pilot selects a master function switch on a control panel and causes multiple subfunctions to become active.

## E. LOCAL EXECUTIVE DETAILED DESIGN AND OPERATION

The DP/M LEX design provides the following control functions for tasks assigned to a PE:

Task execution initiation (scheduling)

Input message processing

Output message processing.

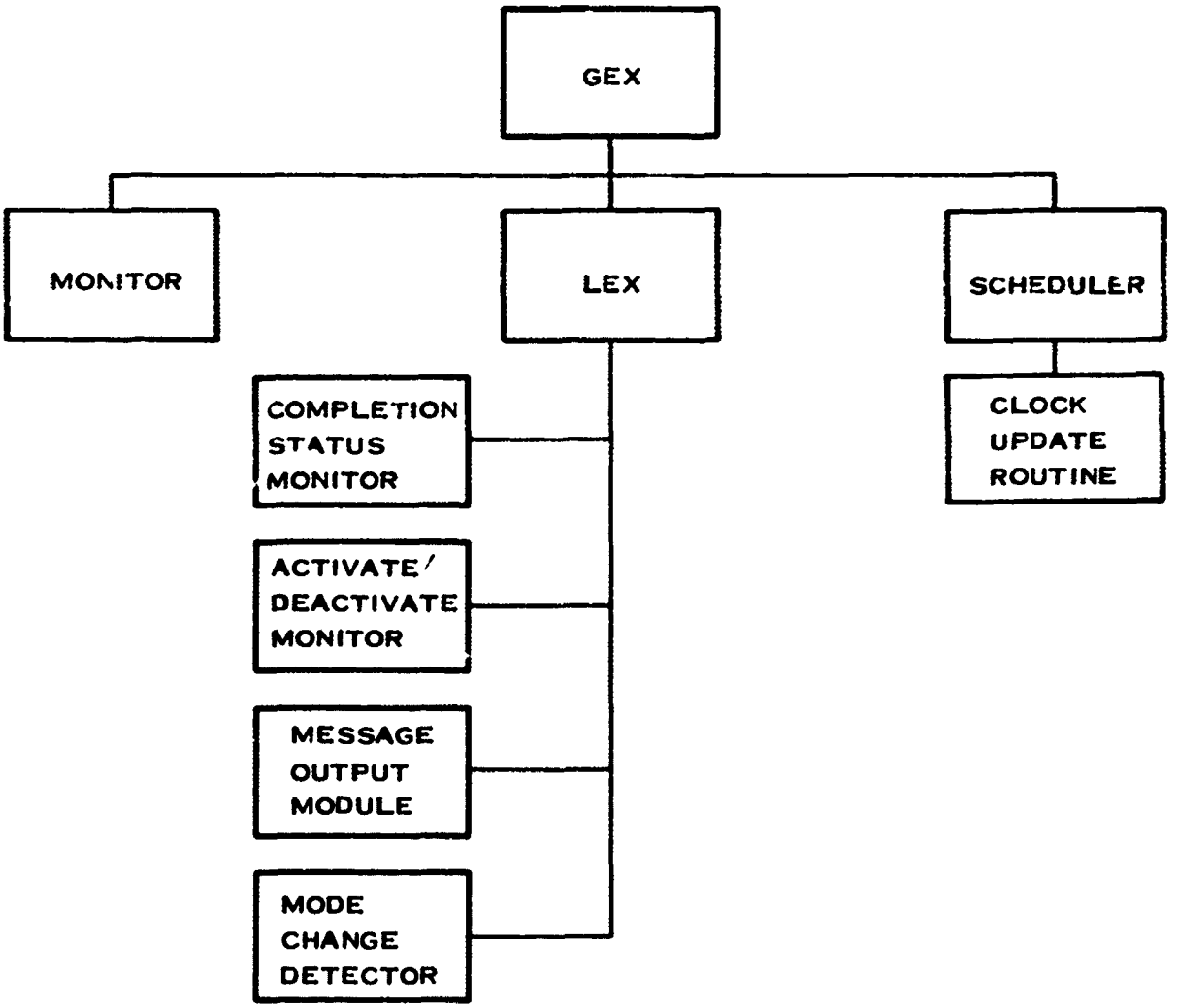



**Figure 141. GEX Block Diagram**

LEX task scheduling is based on satisfactory completion of predecessor conditions for the task. The environment within which the LEX must operate is composed of:

Avionic application tasks

LEX task definition and control tables

PE hardware facilities.

The use of these items by the LEX is discussed in the following paragraphs.

### 1. Application Task LEX Tables

Every avionic application task that is under LEX control can be described via the directed-graph conventions discussed previously. A table structure for these tasks is used to

contain all information necessary for the LEX to control each task. These task preamble tables have descriptive entries as shown in Figure 142. The content of the task preamble table includes:

Task Address: This information is supplied by the linkage editor, giving the address at which execution of the program code for this task is to begin.

Predecessors Remaining: This dynamic entry contains a count of the number of predecessors remaining that must be completed before the task is ready to run. The value of this entry is reset to the total number of predecessor count when this entry is decremented to zero and the task address moved to the dispatcher queue.

Total Number of Predecessors: This entry is static, and indicates the total number of predecessors that are required by this task. This value is used to reset the predecessors remaining entry (described above), when the task is moved to the dispatcher queue.

Total Run Time of Task: The value of this entry indicates the maximum allowable time for this task. The Executive can use this value to monitor the task's time-away from the Executive, by loading the PIT with this value.

Task Iteration Period: This entry is not presently used by the LEX. The entry is reserved for a value which would indicate the repetition period of this task.

Branch Successor Table Pointer: This entry of the preamble table is also unused. However, in keeping with the information content of the rest of the table, this entry points to a list of possible branch successors generated by this task. A branch successor is a task which the currently executing task chooses as its successor from among several branch successors; this decision is made at run time by the currently executing task.

Successor Table Pointer: This entry is a pointer to a table containing the description for a number of tasks which are eligible to run upon completion of this task.

Input Message List Pointer: This entry is a pointer to a table containing the description for all input messages that are to be used by this task.

Output Message List Pointer: This entry is a pointer to a table containing the description for all output messages that are to be output by this task.

A second set of tables, called the Auxiliary Task Preamble Tables (also shown in figure 142), contain additional data for some of the entries in the Primary Task Preamble Table. These descriptive parameters include:

Table A: Table A is pointed to by the branch successor table pointer in the primary task preamble table. It contains a number of entries equal to the number of possible branch successors that the task can have, plus one. The first word contains a count of the number of branch successor entries in this table.

Table B: Table B is pointed to by the successor table pointer in the primary task preamble table. It contains a number of entries equal to the number of successor tasks that are eligible to run, once this task has completed, plus one. The first word contains the count of the number of successor task entries in this table. The successor task entries are pointers to the successor task's preamble tables. Their value is assigned by the linkage editor/loader.

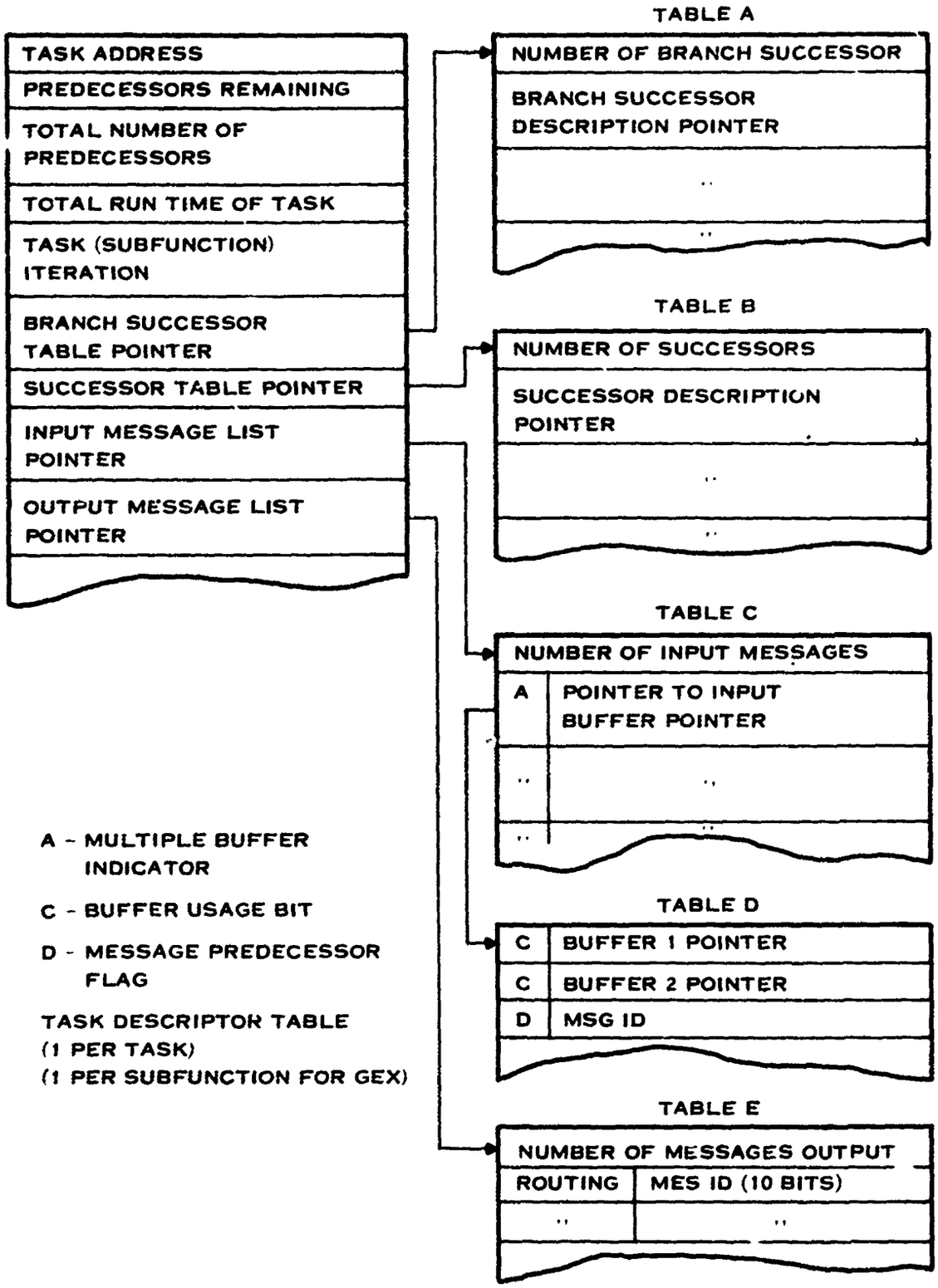



Figure 142. Task Preamble Tables Used by LEX

Table C: Table C is pointed to by the Input Message List Pointer in the primary task preamble table. It contains a number of entries (n) equal to the number of input messages required by this task, plus one. The first word contains the count of the number of input messages. The next (n) entries each represent one of the (n) input messages for this task. Field "A" is the multiple buffer indicator. If "A" is set to logic 1, it indicates that the message has double buffering requirements and the buffers must be used alternately for input data. Buffers are switched when a task that uses data in one buffer is moved to the dispatcher queue. The "Pointer To Input Buffer" entry in Table C points to the first of three possible entries relating to this message in Table D, as described below.

Table D: A set of a maximum of three entries per message in Table D are pointed to by the corresponding message entry in Table C. The first two entries are pointers to the two buffers into which the input message is routed. Field "C" is the buffer usage bit. If set, it indicates that the buffer pointed to by the word in which "C" is set is currently in use. The third word is the ID of the message to which the above information applies. Thus, each input message is described by the following information in the preamble tables: multiple buffer indicator, pointer to an input buffer into which the message is input, buffer usage information, and the message ID. Note that if a message has no double buffering requirement, it is described by two (instead of three) words in Table D (a pointer to the input message buffer, and the message ID).

Table E: Table E is pointed to by the Output Message List Pointer in the (primary) task preamble table. It contains a number of entries (n) equal to the number of output messages generated by this task, plus one. The first word contains a count of the number of output messages. Each one of the other (n) words describes the message generated by the task. The "routing" field indicates the destination of the message: co-resident, local, and/or global bus. The message ID field contains the ID of this message. This information is sufficient for the LEX to route the data to the appropriate user as required.

A summary of the LEX routines that have entries in the task description tables is given in Table 31.

## 2. LEX Usage of Hardware Resources

Several capabilities provided by the PE hardware are used by the LEX. The two primary features are the local and global bus interfaces and the Programmable Interval Timer (PIT). The Bus Interface Unit provides an efficient means of inputting bus messages into the PE with minimum LEX computational overhead. A detailed discussion of this capability was given in Section III, and key features that aid the LEX are now summarized. Figure 143 describes and shows the software input message scheme used by the LEX.

The aim of the input message processing is to efficiently correlate input data messages with those recipient tasks that require this data. Each message broadcast on one of the communication buses is composed of a logical message identification (ID) and the data in the message. The use of logical message IDs permits the data to be broadcast to any number of PEs connected to a bus. Any PE desiring a given set of data in a bus message sets a bit in its Message Identify Associative Match Map (MIAMM) that corresponds to the logical message ID. The PE bus interface also uses a Message Buffer Vector Space (MBVS) which contains pointers to buffer

**TABLE 31. LEX PREAMBLE ENTRY DESCRIPTIONS**

| Table Entry | Users | Description |
|---|---|---|
| Task Address | Dispatcher | Entry point address of task code. |
| Predecessors Remaining | LEX Scheduler | Number of predecessors remaining. When zero, task is ready to move to dispatcher table. |
| Total Number of Predecessors | LEX Scheduler | Number of different predecessors required for this task. |
| Total Run Time of Task | Dispatcher | Maximum time to execute this task. |
| Task Iteration Period | Unused | Period of repetition of this task. |
| Branch Successor Table Pointer | Unused | Pointer to a table containing pointers to the preambles of run time branch successors of task. |
| Successor Table Pointer | Successor Scanner of LEX Scheduler | Pointer to a table containing pointers to the preambles of successors of this task. |
| Input Message List Pointer | Service Module, Input Message Scanner | Pointer to a descriptor list for messages input to this task. |
| Output Message List Pointer | Output Message Interpretor | Pointer to a descriptor list of messages output by this task. |
| Time-Ordered List Pointer | GEX | Pointer to this task's descriptor list in the time-ordered linked list. |
| Table A | Unused | Contains count of number of possible run time branch successors of task and pointers to task preambles for these tasks. |
| Table B | Successor Scanner | Contains count of number of successors of this task and pointers to the preambles for these successors. |
| Table C | Input Message Scanner, Service Module | Contains count of number of messages input to task. Flag A indicates if message requires multiple buffers. Flag B indicates multiple receipt of same message per one iteration of the recipient task. |
| Table D | Input Message Scanner, Service Module | Contains pointers to alternate buffers used by an incoming message. Flag C indicates currently active buffer. |
| Table E | Output Message Interpreter | Contains count of number of messages output by task and routing information for each message. |

areas in memory where input message data is to be written. Addressing of the MBVS occurs as follows: Each incoming message ID that has a bit set in the MIAMM is to have the message data stored in PE memory. Once this occurs, the message ID (value 0-1023) is appended to the address of the first word in the MBVS to form a pointer address to an entry in the MBVS. This entry is itself a pointer to a data buffer area in which the incoming message is to be stored. Three items are found in the buffer area. The buffer address from the MBVS points the hardware to the buffer space as shown in Figure 143. The hardware uses the length (m) to count

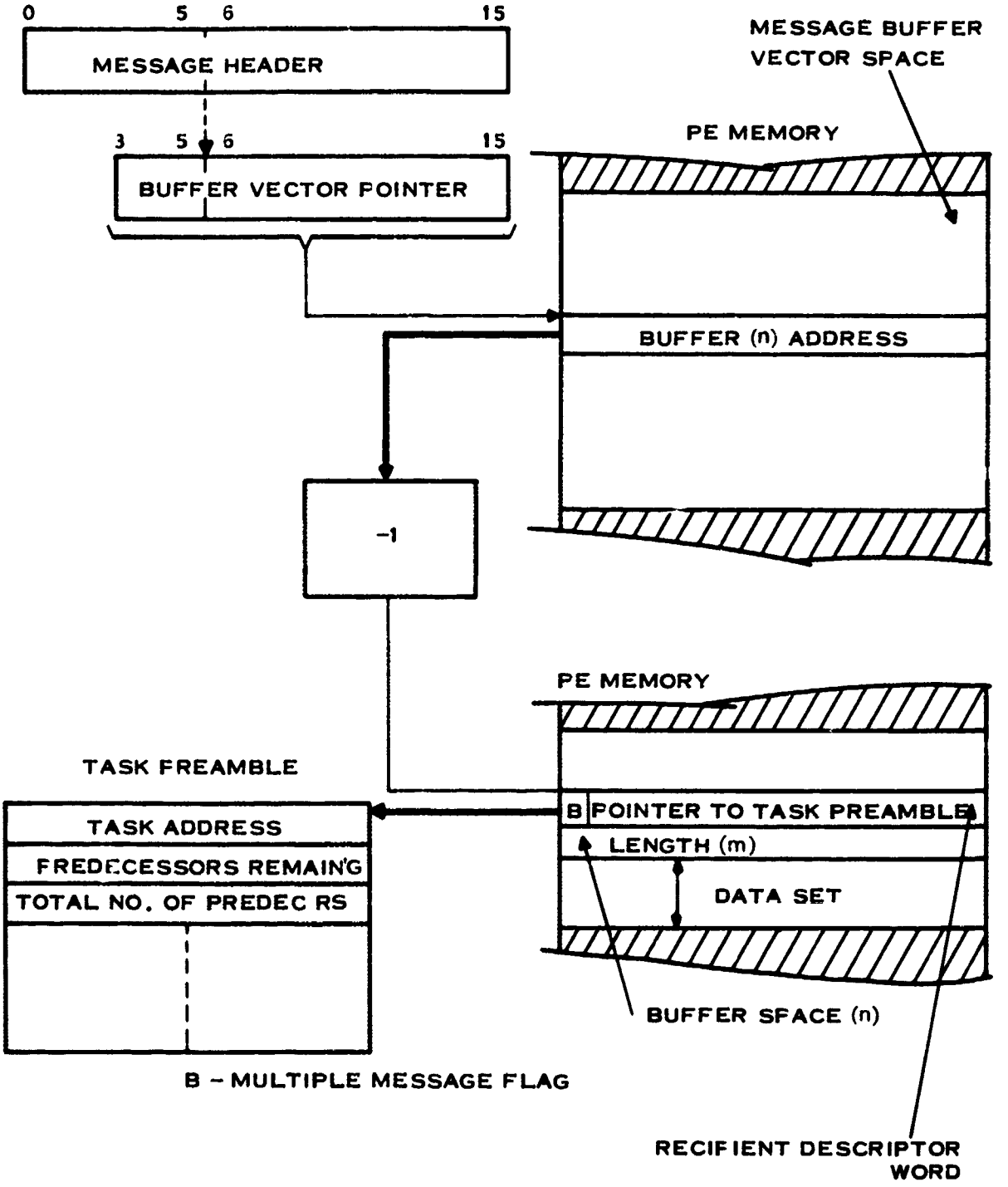



Figure 143. Software Input Message Correlation Scheme

incoming data words into the buffer. This length data is initially preassigned for all appropriate buffer locations. The word preceding the length word in the buffer space contains a pointer to the preamble of the task that uses data from this buffer. If this word is zero, it indicates that the message that has just been received is part of a common data pool, and has no successors. Field B is the multiple-message flag; if set, "B" indicates that this message has been previously received, and every subsequent receipt of this same message shall not be allowed to decrement the recipient task's predecessor count. "B" is reset when the task is moved to the LEX dispatcher queue.

The other primary hardware feature of the bus interface used by the LEX is the BIU Input Queue Pointer (IQP) discussed in Section III. Each bus interface maintains an eight-word circular queue in dedicated segments of read/write memory that is used to store the header (status and ID) information for each message received by the PE. The IQP contains an address of the next entry available in the queue (i.e., "IQP-1" is the address of the last complete message received). The LEX uses the IQP to process input data required for scheduling tasks within the PE.

The two primary non-bus-oriented PE hardware facilities used by the LEX are the hardware vectoring of interrupts and the PIT. The PE interrupt structure provides for the honoring of armed interrupts by the automatic vectoring of the processor program counter to an address associated with the given interrupt. This vector address contains the derived operand for an Exchange Status and Program Counter instruction (i.e., a new program counter and status word). This new status information is loaded into the proper PE hardware registers after the program counter and status word have been saved. The PIT is a hardware loadable/readable real-time clock (or counter) that can be used by the LEX for timing purposes such as ensuring a task executes within an allowable time period.

## 3. Detailed LEX Module Description

The DP/M LEX is composed of the following functional modules:

- Bus Interrupt Service Routines
- The Task Scheduler
- The Dispatcher
- The Output-Message Interpreter Module
- The Message Transmitter Modules.

A detailed discussion of the operation of these routines follows.

### *a. Bus Interrupt Service Routines*

Normal bus interrupt service routines for both the Global and Local buses are invoked when an incoming or outgoing message completes transfer on its respective bus, and the proper interrupt has been enabled. The interrupt trap locations initially contain the address of the service routine and a status word which defines the state of the other interrupts (masked or unmasked). When the normal bus interrupt occurs, PE hardware saves the current Program Counter (PC) and Status Word (SW) in the interrupt stack, uses the new status word from the trap location for machine status, and branches to the Bus Interrupt Service Routine as pointed to by the vector in the trap location.

The Interrupt Service Routine saves all registers, inputs the interrupt status word, determines whether this interrupt was caused by an incoming or outgoing message, and transfers control to either the scheduler or the message transmitter. In the normal task processing mode, the input message complete interrupt is disabled and is enabled only when a PE is in the background mode. The term background mode is used within the context of the DP/M to represent that time in which the PE is not actively executing application tasks. Some discussions refer to this period as idle time; however, such time can be used for low-priority interruptible tasks. Possible background tasks include execution of performance assurance or diagnostic test routines. Any high-priority activity such as receipt of an input message for a task can interrupt and pre-empt execution of background mode tasks with no resultant degradation in system performance.

### *b. Task Scheduler*

The Task Scheduler is made up of five sub-modules which service different aspects of the LEX scheduling function. The Task Scheduler is responsible for determining which tasks in a PE have their predecessor requirements satisfied and are, hence, ready to be given control. The Task Scheduler consists of the Task Completion Entry Point (TCEP), Branch Successor Scanner (BSSCAN), Successor Scanner (SSCAN), Input Message Scanner (IMSCAN) and the Scheduler Service Module (DISPIN). The interrelationship between these routines is shown in Figure 144.

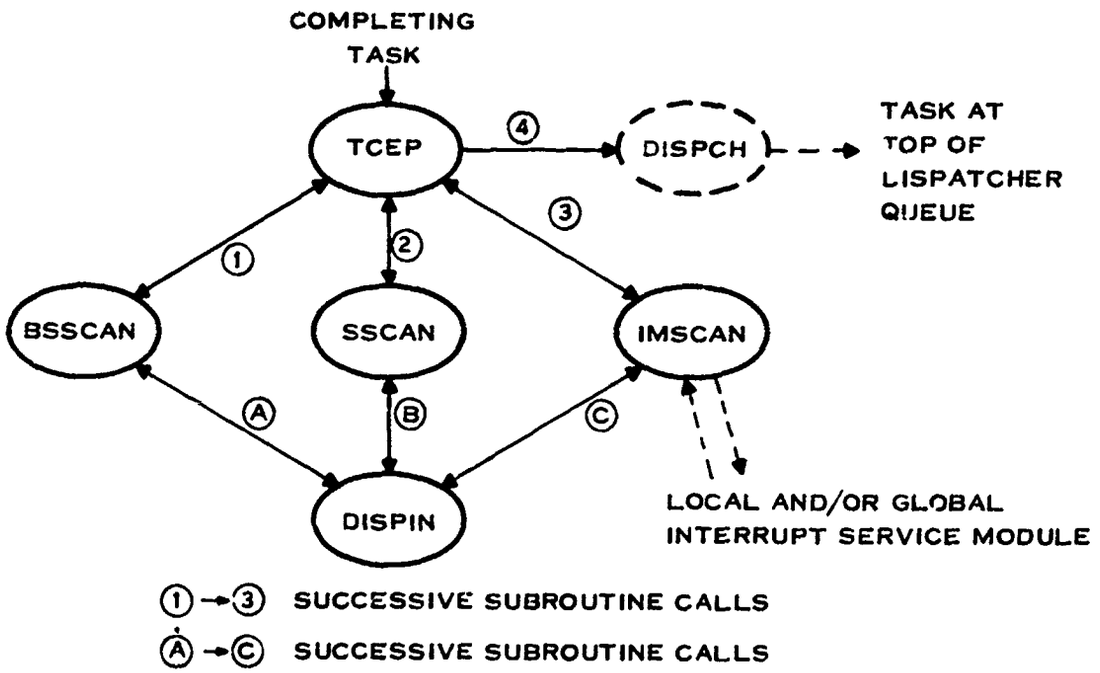



**Figure 144. Task Scheduler Subroutine Interrelationships**

(1) Task Completion Entry Point (TCEP)

The Task Completion Entry Point segment of the scheduler is called by a completing task that is returning control to the LEX. The task passes its task identification (ID), task status, and branch successor information to the LEX. The TCEP disables the PIT (which was armed by the Dispatcher module) and saves the information that has been returned by the task. It calls the Error Monitor routine if the task had returned bad status information, (e.g., the data it had computed was not valid, a device I/O error parity occurred, etc.). The TCEP is the "main" scheduler routine with the BSSCAN, SSCAN, and IMSCAN sub-modules called from it, and returning control to it. The TCEP exits to the Dispatcher after IMSCAN returns control to it. The task ID returned to the TCEP is in the form of a pointer to the preamble table of the returning task. The branch successor ID is similarly translated into a pointer to the branch successor task's preamble table. Thus, LEX modules needing this information are able to use it directly without further computation.

(2) Branch Successor Scanner (BSSCAN)

The BSSCAN is called from the TCEP. Its purpose is to examine the branch successor data returned by the completing task to TCEP and take appropriate action. This action includes examining the Branch Successor Word flag that has been returned by the task. If a zero flag is returned, the completing task has no branch successors, and BSSCAN returns to the TCEP. If the flag is non-zero, the completed task did have a branch successor, and the flag is a pointer to the preamble table of the branch successor. BSSCAN uses this pointer to retrieve the value of the "number of predecessors remaining" entry for the branch successor from the preamble table and it decrements this value. If the value continues to be non-zero, it is stored back into its location in the preamble table. If it decrements to zero, BSSCAN calls the Scheduler Service Module. BSSCAN then makes a subroutine call to the TCEP.

(3) Successor Scanner (SSCAN)

When the BSSCAN returns to the TCEP, the TCEP then transfers control to the SSCAN. The SSCAN determines the successors that are affected by the completing task and takes appropriate action. A task's successor information is found in the completing task's preamble table. An entry in the preamble table points to an auxiliary preamble table which contains the successor information (LEX Auxiliary Task Preamble Table). Each successor in the completion task's table is represented as a pointer to each successor task's preamble table. Thus, SSCAN can directly access information for any of the successor tasks' preamble tables without any further computations.

SSCAN uses the pointer to the completing task's preamble returned by the completing task to find a pointer to the successor table (Table B of the Task Preamble Table). If the pointer is zero, this indicates the task has no successors, and SSCAN returns control to TCEP. If the pointer is non-zero, SSCAN obtains the successors preamble address from the table pointed to. Using the preamble address, SSCAN obtains this successor's predecessor count (NPRZ). SSCAN then decrements NPRZ and, if the count decrements to zero, SSCAN calls the scheduler service module. If NPRZ is still non-zero, the value will be stored back, and SSCAN looks for the next successor, and continues until all successors have been serviced. SSCAN then returns control to the TCEP.

(4) Input Message Scanner (IMSCAN)

The Input Message Scanner examines the FIFO input message queues (IMQ) for messages input from the Global and/or Local bus. It establishes a recipient for each input message and performs necessary table update actions. It has an entry point from TCEP as well as entry points from the normal Local and Global bus interrupt service routines.

The flow diagram of IMSCAN is shown in Figure 145. When IMSCAN is entered from TCEP, the Global bus interface status is examined to see if any messages have been received since the last execution of IMSCAN. If it is determined that no messages have arrived, the Local bus interface status is examined for similar activity. IMSCAN exists to TCEP if no messages have come in over either bus. The following paragraph describes the action of IMSCAN when it detects message(s) in the global IMQ. The action of IMSCAN is similar when it detects message(s) in the local IMQ.

The first operation of the IMSCAN is to obtain the Input Queue Pointer (IQP) from the hardware interface. The hardware IQP contains an address to a list of message headers of bus data received and stored into PE memory. To facilitate asynchronous operation of bus input message processing and avionic application code execution, a software queue pointer is also maintained by the IMSCAN module. While a provision does exist for a high-priority message to interrupt the PE to gain rapid servicing, the normal mode of input-message processing is non-pre-emptive and is accomplished during time periods between application task execution. After checking for queue overflow, the pointer returned from the interrogation of the IQP and the software maintained current input queue pointers are compared. If determined equal, global input message processing is complete and the program begins examining the local input message queue.

If the pointers are not equal, there are message IDs still posted in the queue that must be serviced. The message ID is extracted from the location pointed to, and the location is zeroed to signify processing is completed. The message ID is then appended to a fixed-base address. This new address vector points to a location in PE memory which contains the address of the buffer pointer to obtain a recipient descriptor word (pointer to task preamble) associated with this message ID. This descriptor word is placed in this location at system initialization and can be changed only by the LEX. The message descriptor word is made up of two fields: (1) the most significant bit (MSB) which, if set to one, indicates that this message has already arrived at least once into the PE, and (2) an address which points to the recipient task's preamble table. If this word is zero, the message has no successors, and IMSCAN services the next message. Messages that have no successors are generally common data arriving for use by several tasks that have iteration rates much faster than the subfunction using this data.

If the MSB of the recipient descriptor word is set it indicates that the message has been used previously to decrement the recipient task's predecessor count; the subsequent arrival of this same message leaves the recipient task's predecessor count unaffected. If the MSB of the recipient descriptor word is not set, IMSCAN sets the bit and obtains the predecessor count (NPRZ) of the recipient task by using the pointer to this task's preamble table from the recipient descriptor word. IMSCAN then decrements this predecessor count. If NPRZ goes to zero, IMSCAN calls the Scheduler Service Module to take appropriate action; if NPRZ continues to be non-zero, this updated value is restored into the appropriate table in the task's preamble table, and IMSCAN loops back to check for more messages. When no more messages are found, IMSCAN begins to process the Local bus input queue in the same manner as described above.

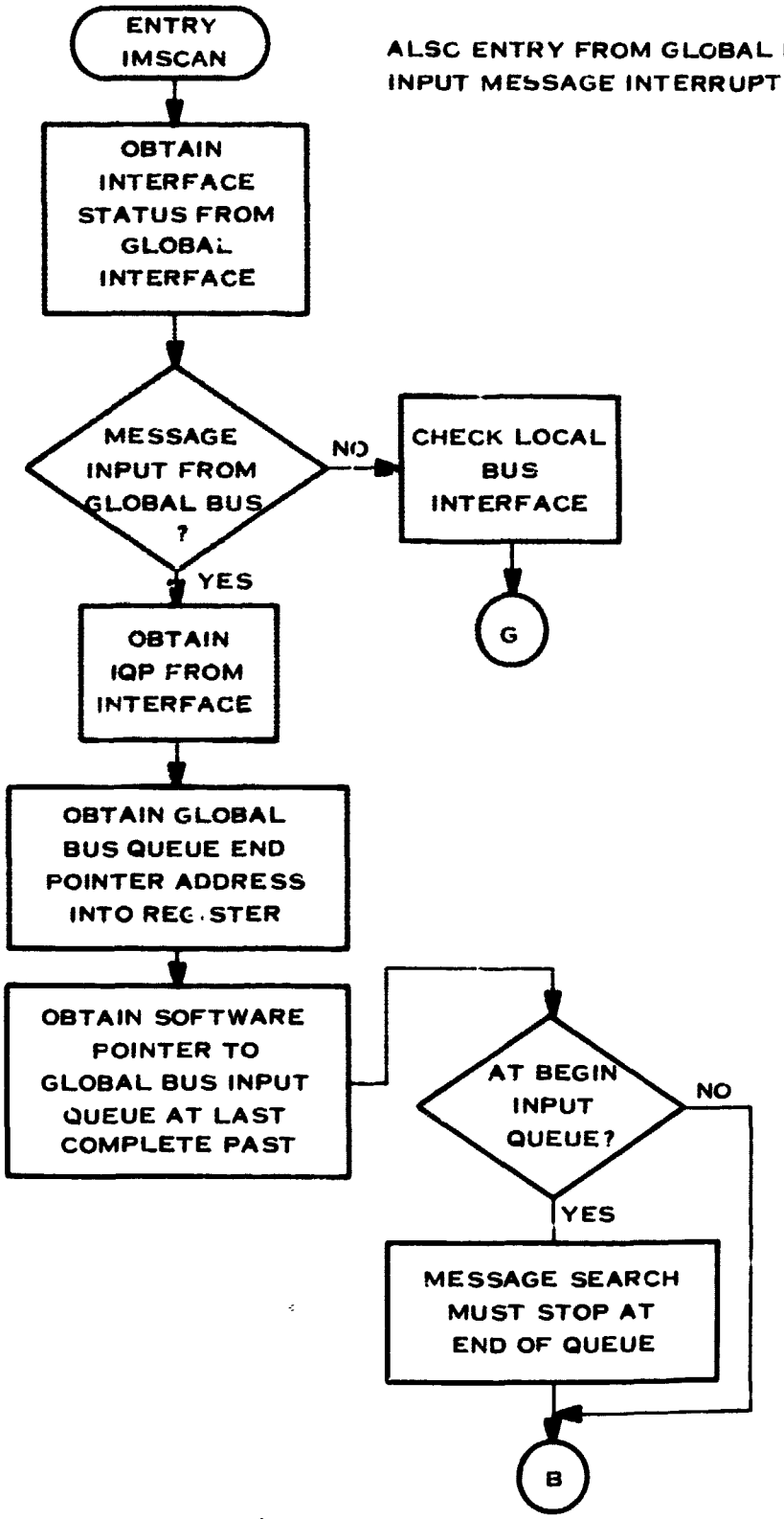



Figure 145. Task Scheduler Message Scanner (Sheet 1 of 5)

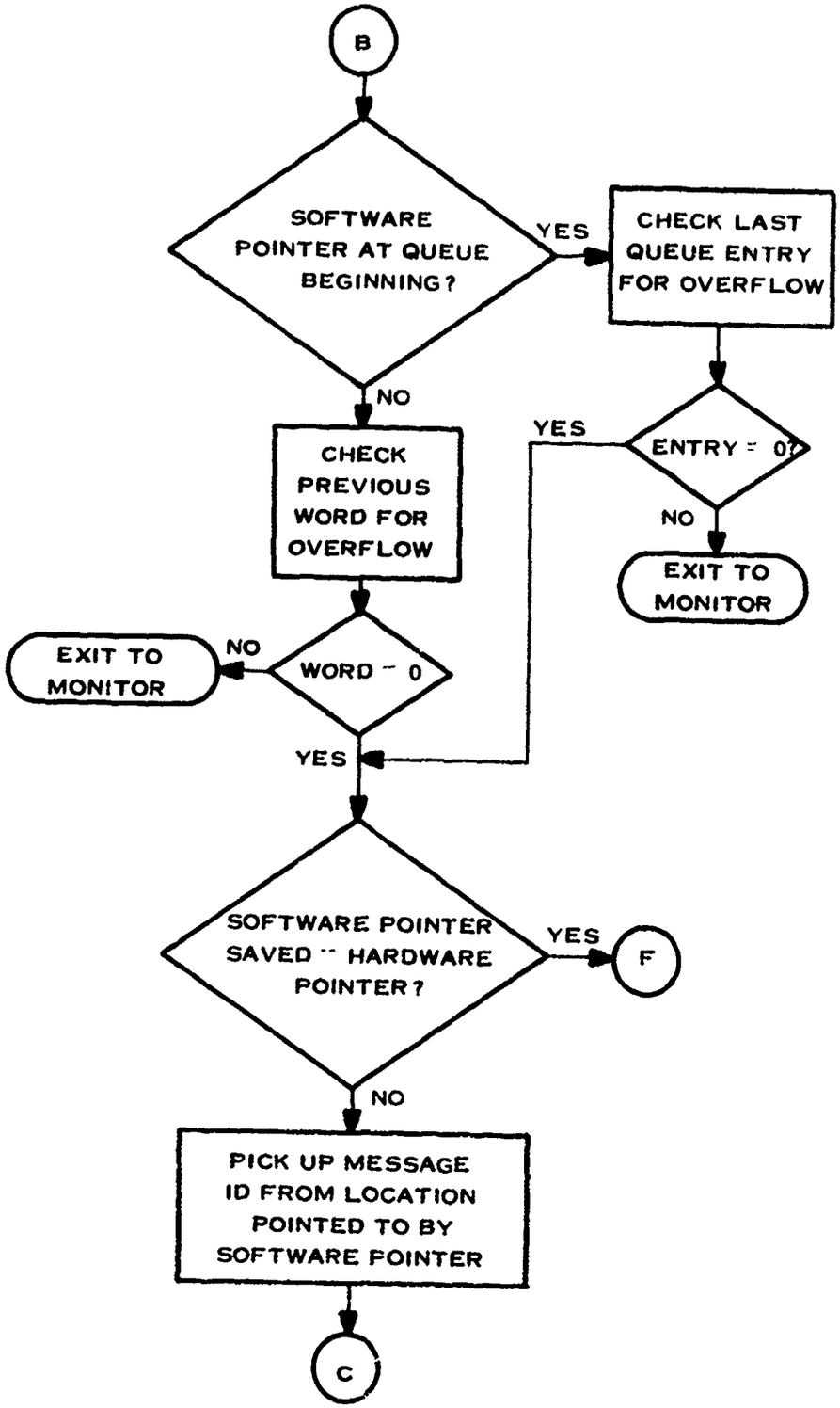



Figure 145. Task Scheduler Message Scanner (Sheet 2 of 5)

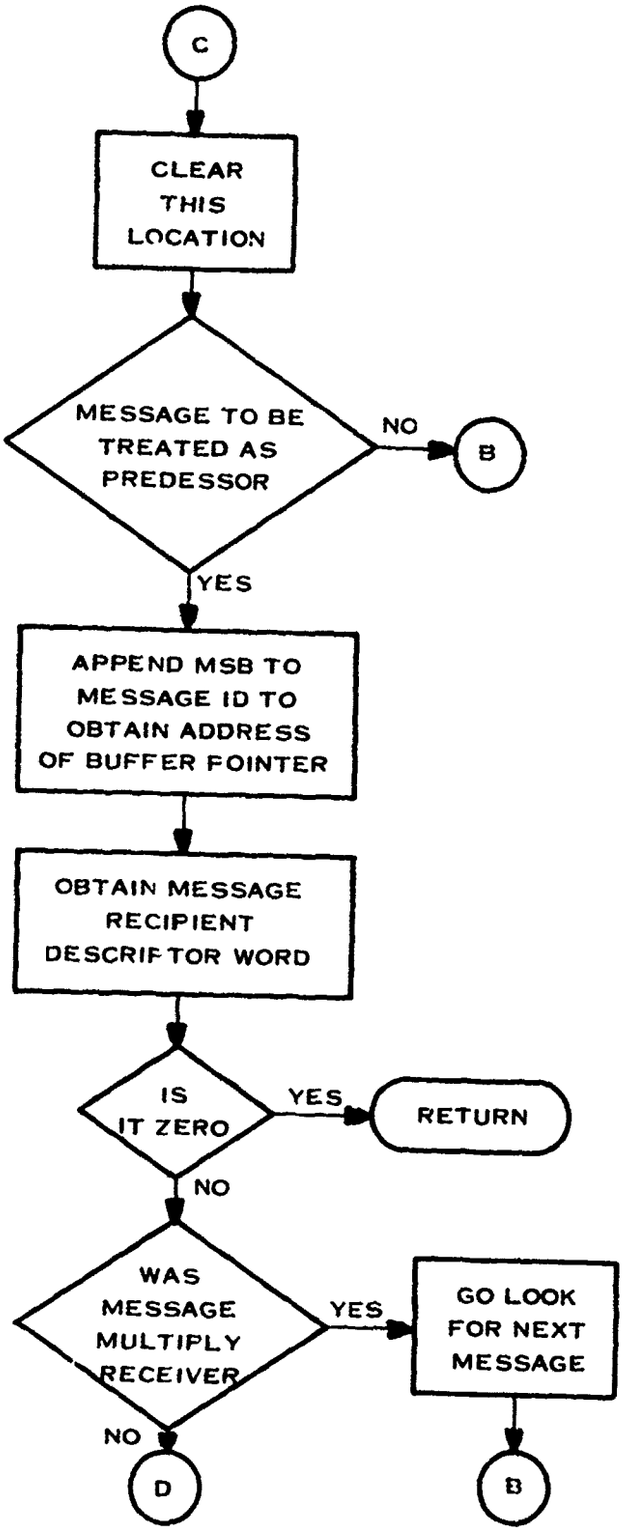



Figure 145. Task Scheduler Message Scanner (Sheet 3 of 5)

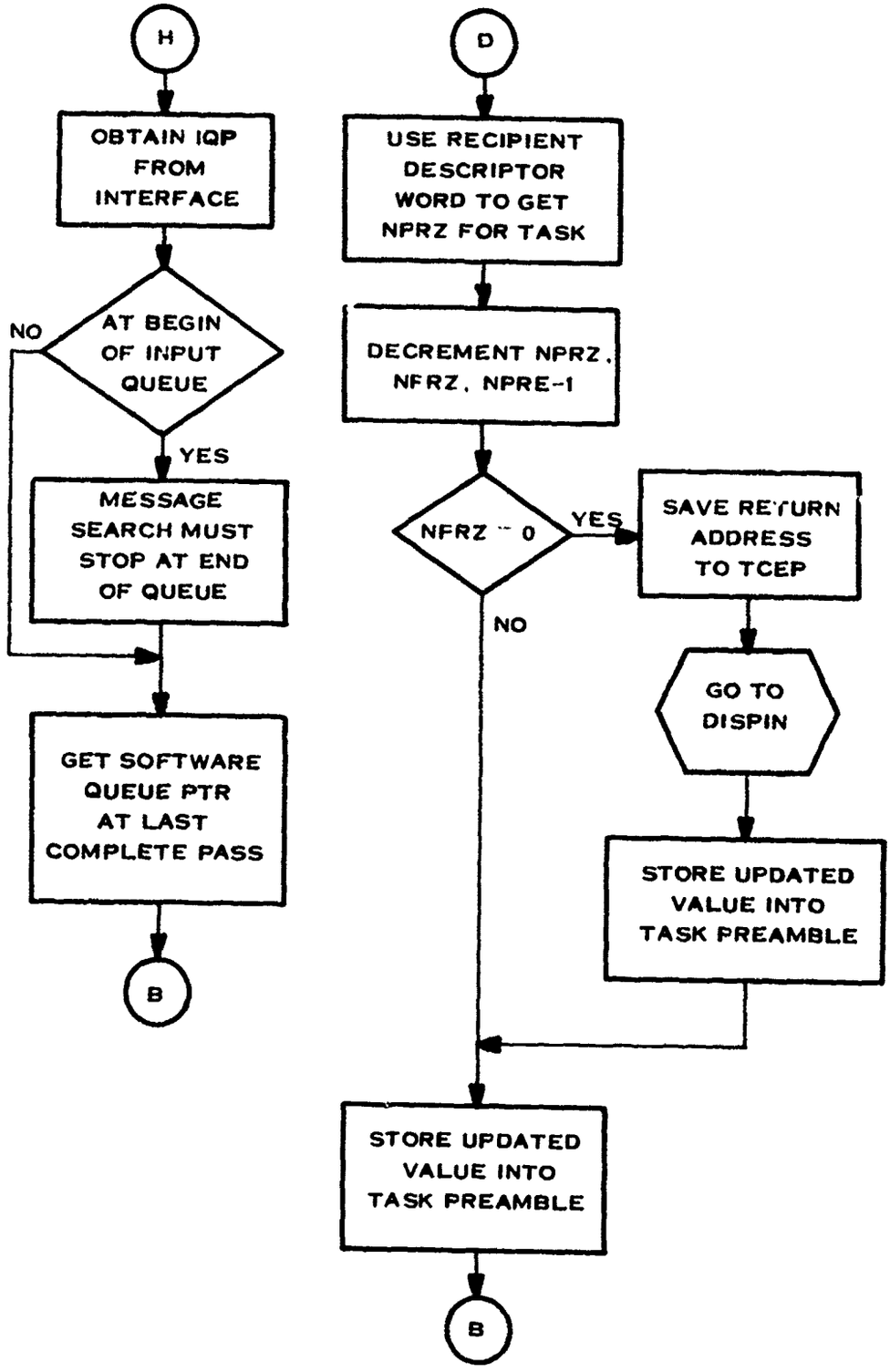



Figure 145. Task Scheduler Message Scanner (Sheet 4 of 5)

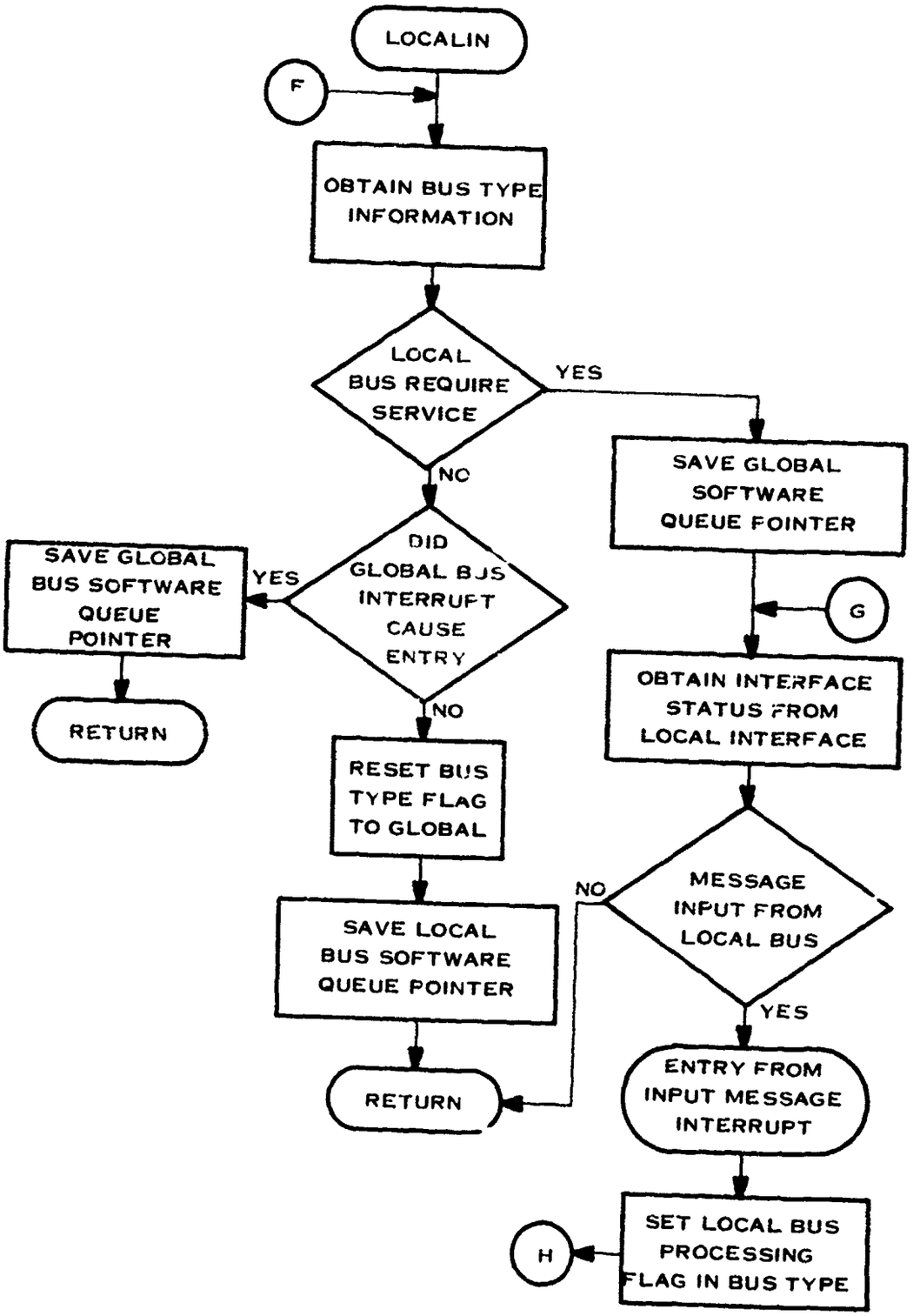



Figure 145. Task Scheduler Message Scanner (Sheet 5 of 5)

After input queue processing is complete, the IMSCAN returns control to TCEP, if it was entered from TCEP. Otherwise, it returns to the processing in progress at the point it was interrupted.

(5) Scheduler Service Module (DISPIN)

The Scheduler Service Module is called by all other task scheduler submodules (except TCEP) when they have determined that a task is ready to be moved to the dispatcher queue. The function of the DISPIN is to move the ready task's preamble address to the dispatcher queue, reset control flags, reset the ready task's predecessor count, and provide an alternate input buffer for data routed to this task if required, provided all data input into the current buffer is complete.

The flow graph of DISPIN is shown in Figure 146. The calling program provides DISPIN with the preamble address of the task requiring service. Using this information, DISPIN obtains the pointer to the list of input messages associated with this task. If the input message table pointer is zero, DISPIN recognizes that the task requires no more input message service and proceeds to move pertinent task information to the dispatcher table. If the pointer is non-zero, DISPIN obtains the message count from the location pointed to by the table pointer and information about the first message from the next location (see Figure 142).

DISPIN then determines if the message has a requirement for double buffers. A requirement for double buffering of input message occurs if the task using the message data does not process all the data in the input buffer before the message is likely to be received a second time. If it determines that the message requires only one buffer, it proceeds to find out if the message is merely being used by the task or is also a predecessor. If it is not a predecessor, DISPIN takes no further action pertaining to this message, but proceeds to service the next message. If the message is a predecessor, DISPIN clears the multiple-message flag in the recipient descriptor word (Figure 146) and then starts to service the next message.

When a message has double-buffering requirements, DISPIN's decision making becomes a little more complex. First, it must determine which buffer is currently in use, whether data input to the currently used buffer is complete, and then swap buffers accordingly. From this point on, its logic is identical to that of the single buffer case as described above. It determines if the message in question is merely being used by the task or is a predecessor. If it is a predecessor, DISPIN clears the multiple-message flag as described above, otherwise, it loops back to process the next message.

When all messages have been processed by DISPIN, it proceeds to move the address of the task being currently serviced into the dispatcher queue. The current dispatcher queue position pointer is obtained first, and the task's preamble table pointer which was supplied by the calling program is put into the dispatcher queue. The number of predecessor locations (NPRZ) is the reset to the total predecessors for this task. DISPIN supplies this information in a register to the calling program. When DISPIN has posted the task's preamble address into the dispatcher queue, it returns to the calling program.

### *c. Dispatcher*

The Dispatcher is called from the TCEP. The Dispatcher transfers control to the task which resides at the top of the Dispatcher queue and provides the task with a pointer to a list of

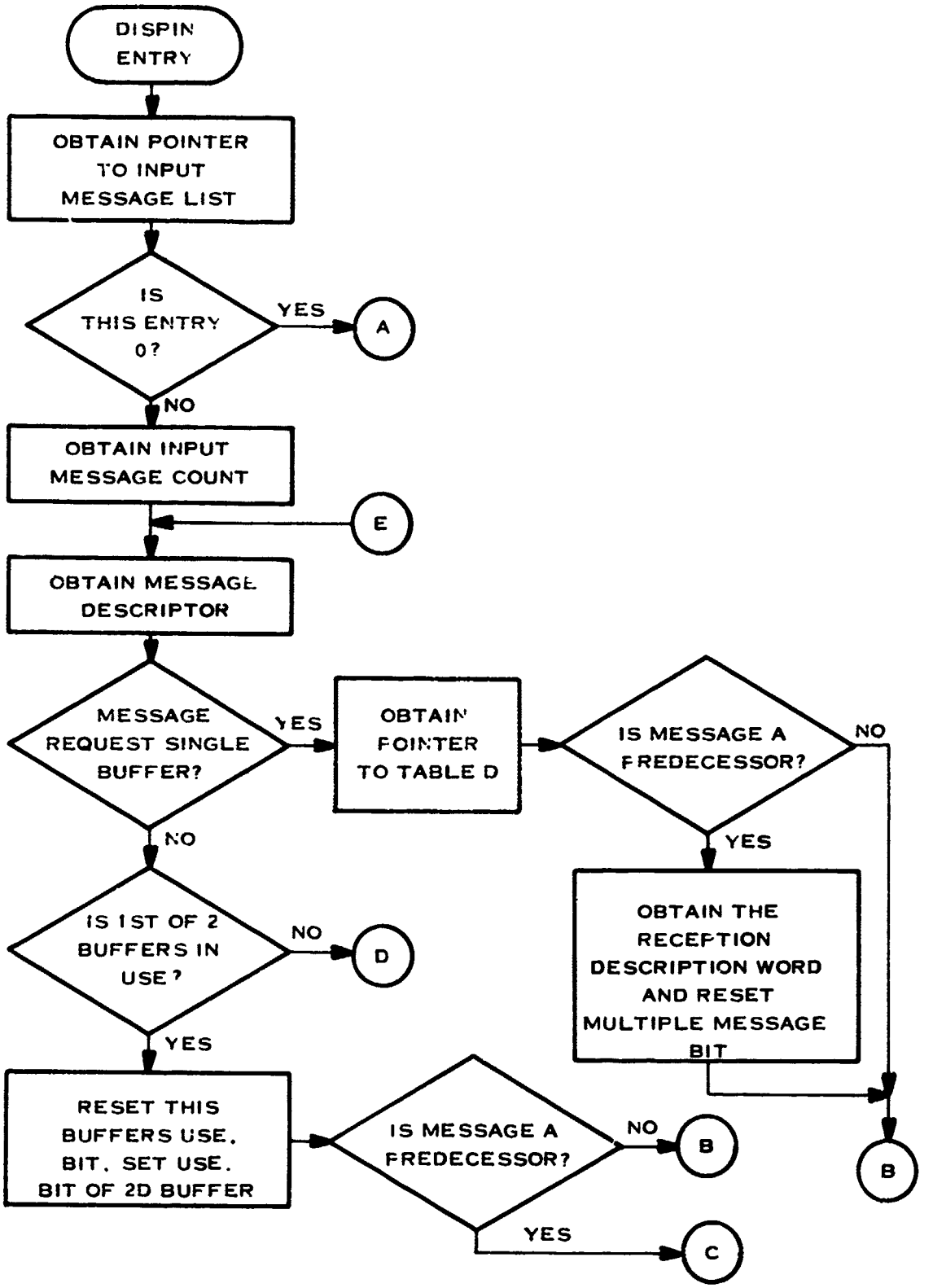



Figure 146. Task Scheduler Service Module (Sheet 1 of 3)

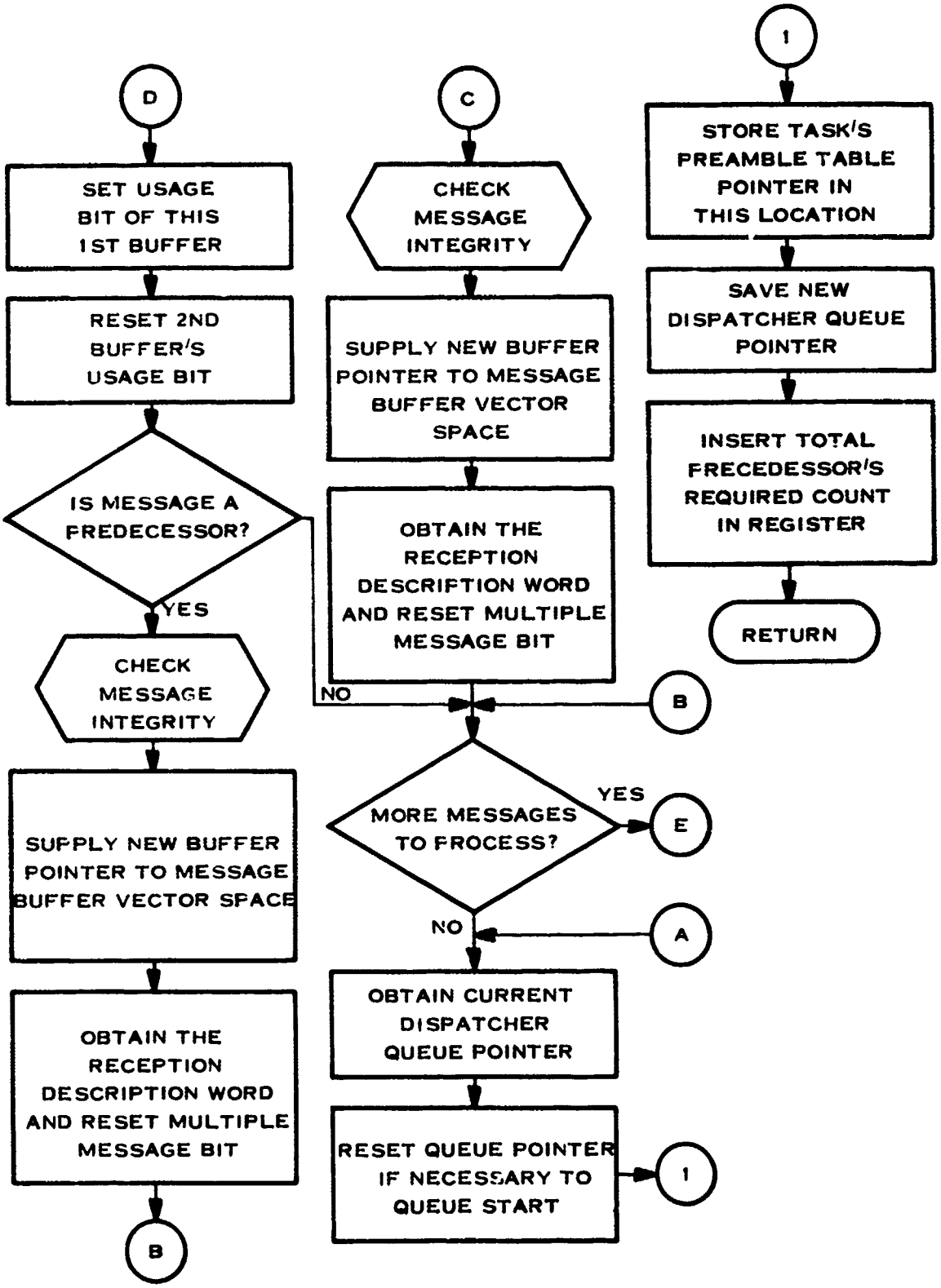



Figure 146. Task Scheduler Service Module (Sheet 2 of 3)

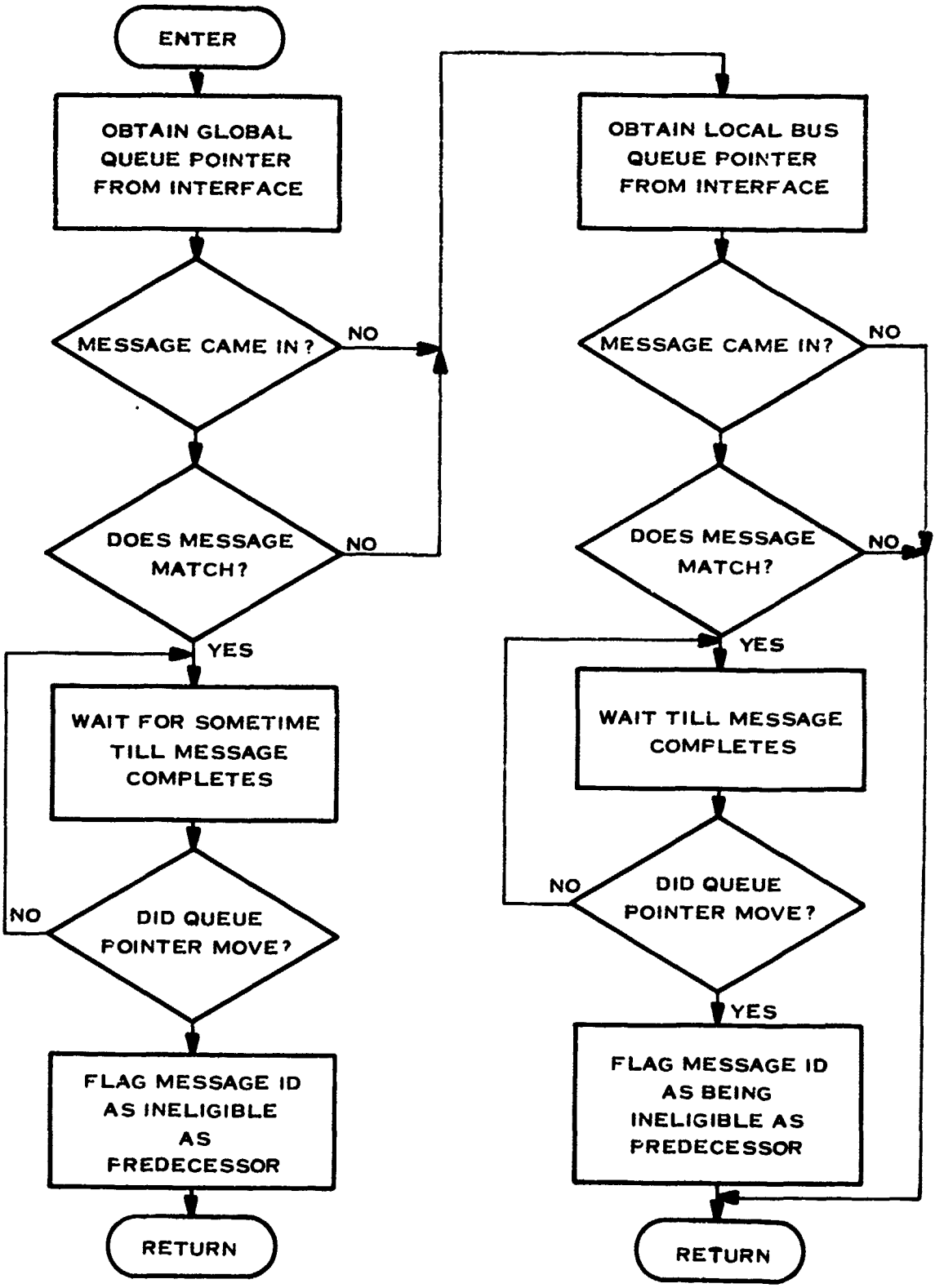



Figure 146. Task Scheduler Service Module (Sheet 3 of 3)

required input message buffers. It also loads the PIT with the maximum execution time for the task immediately before transferring control to the task: this enables the LEX to monitor the task's time away from the Executive. The Dispatcher obtains the current location of its queue that it must service from a fixed location in PE memory. It then examines the contents of this location. If zero, the LEX has nothing to run, and control is transferred to a background mode program such as a PE built-in-test routine. If the Dispatcher does find an entry in this location, it saves the entry and the new dispatcher queue position pointer. It then builds a list of message pointers, to messages that are required by task, and provides a list pointer to the task. Finally, it loads the PIT with the execution time (from the task's preamble table), and transfers control to the task.

### *d. Output Message Interpreter Module*

The Output Message Interpreter Module (OMRQST) of the LEX is called by a task when it needs output mesage service. When tasks generate outputs for other tasks, information about the physical destination of these outputs is not required by the message generating task. OMRQST obtains logical header ID information pertaining to the output messages generated by the task from the task's preamble table, and accordingly transmits output message over the Local bus, the Global bus, or both. OMRQST takes no action for messages that are for PE co-resident tasks other than return control to the task.

The task calls OMRQST when it has a data set ready to output a message. It supplies the following information to the OMRQST:

Requester task's ID (in the form of the task's preamble pointer)

Buffer pointer to the output message buffer

Message ID.

OMRQST then obtains the pointer to the output message list by using the preamble pointer returned by the task. From the output message list, OMRQST decodes routing information for the message that needs to be output. Should the task return a message ID of zero, the OMRQST decodes this as being a request to output all message that are generated by the requesting task.

After identifying the proper destination bus routing, OMRQST determines if the desired bus interface is available for data output. If the interface is available, the output message buffer pointer is sent to that interface to initialize an autonomous output transfer. If the interface is busy, the address of the output buffer is posted into a software FIFO output queue. Then, dependent upon whether the task had requested the output of a specific message or of all messages, OMRQST returns or loops back to process any additional messages for the task that must be output. After OMRQST has completed processing, it returns control to the calling task.

### *e. Message Transmitter Modules*

The Message Transmitter Modules (MSGXMT) (one each for the Local and the Global buses), are invoked upon receipt of the output message complete interrupt from the Local and the Global buses. Two identical MSGXMT modules are provided to service each of the two interrupts. Their only difference is the use of different output message queues and different interface I/O commands.

When an output message transfer completes, the interrupt vector which points to the message transmitter module in the interrupt trap location of the output complete interrupt is automatically loaded into the program counter of the PE. The Bus Interrupt Service module is invoked which then transfers control to MSGXMT, examines the appropriate output queue, obtains the buffer pointer, zeros the location, updates the queue pointer, and outputs the buffer pointer to the appropriate interface. An autonomous output message trnasmission is thus initiated. At this point MSGXMT returns to the PE program which was interrupted when the output message completed.

## F. DETAILED GEX PROGRAM DESCRIPTION

The Global Executive modules will most likely reside in one or two designated PEs of the DP/M system. The GEX schedules time-dependent subfunctions in the DP/M system. The GEX generates a "go" message to a subfunction when it is time to run that subfunction and other predecessor conditions have been satisfied.

### 1. Global Executive Data Base and Hardware Usage

As with the LEX control structure, the Global Executive uses a data base made up of tables which contain information required to correctly initiate and control subfunctions that are scheduled by the GEX and initiate and control "tasks" which reside in the GEX PE. Information pertaining to the active/deactive status of subfunctions, completion messages, subfunction iteration periods, output messages, predecessor-successor relationships among tasks, etc., is contained in these tables. Two data structures are defined for use by the GEX: a subfunction preamble table and a time-ordered link list.

#### *a. Subfunction Preamble Table*

The format of the subfunction preamble tables is identical to the LEX task preamble tables described previously. Most entries which pertained to a task also pertain to subfunctions in this discussion The subfunction preamble table format is shown in Figure 147, with the exception of a pointer to the task's time-ordered list. Task and subfunction preamble tables located in the GEX PE have one additional entry: the pointer to the task's time-ordered list. Tasks or subfunctions which are time-scheduled by the GEX are described by time-ordered linked lists in addition to their preamble tables. The pointer that is being discussed in this paragraph points to this task's time-ordered list. If the task (or subfunction) is not time-scheduled, this entry is zero.

#### *b. Time-Ordered Linked List*

All subfunctions in the DP/M system which are time-scheduled by the GEX scheduler are represented by data tables which are elements of a time-ordered linked list (i.e., list entries are arranged with first low values followed by successively higher values). The words of this data table (shown in Figure 148), when viewed collectively, contain information necessary to control the scheduling of the time-scheduled subfunctions in the system. Entries in these data tables include:

Clock Update Task Flag: If this word is non-zero, this data item contains information pertaining to the clock-update task. The clock-update task which is a periodically scheduled GEX module is described in a subsequent paragraph.

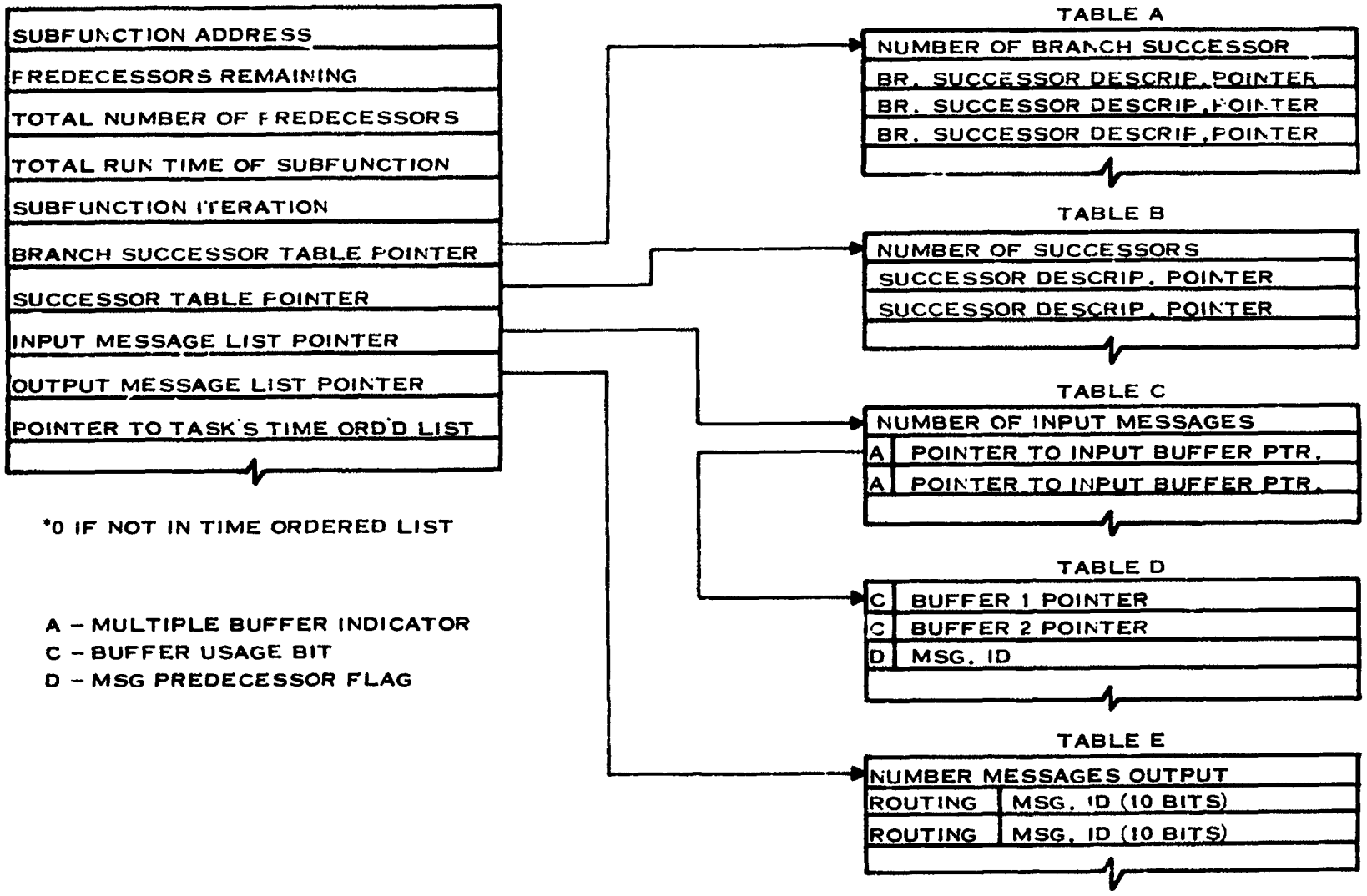



Figure 147. Subfunction Preamble Tables Used By GEX

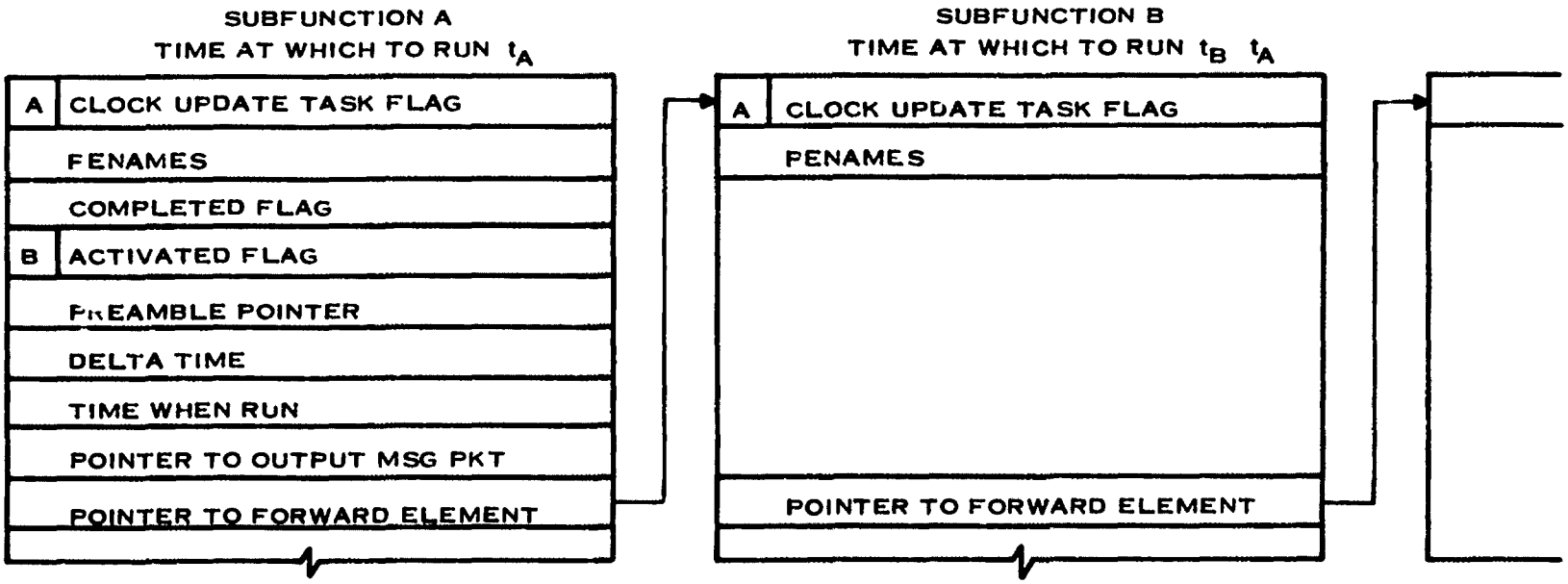



Figure 148. Time-Ordered Linked List For GEX Scheduler

Mode Flag: This flag, if set, indicates that this subfunction requires mode information (e.g., NARBS, RADAR, NAV, etc.). It is used by the mode change detector module of the GEX to post information into the "go" output message that is sent by the GEX to the subfunction.

PENAMES: The PENAMES word in the time-ordered linked list is bit-oriented. Each bit (right to left) represents the number of the PE (1 to 16) in which the starting node or nodes of the subfunction is located. While the PENAMES word is presently shown as one word, multiple words can be used, depending upon actual system requirements.

Completed Flag: The completed flag is a dynamic entry in the time-ordered list. It is set to indicate one of three conditions: (1) the subfunction associated with the table completed successfully previously, (2) the subfunction associated with this table did not complete, (3) the subfunction associated with this table has been deactivated.

Activated Flag: The activated flag is a dynamic entry in the time-ordered list. It is set to logic "1" when a subfunction is considered "active" and eligible for being sent a "go" message by the GEX.

Preamble Pointer: The preamble pointer is a static entry in the table. It points to the preamble table associated with this subfunction.

Delta Time: The Delta Time is a static entry in the time-ordered table. Delta Time indicates the iteration period of this subfunction.

Time When Run: This entry is dynamic. When it is time for a subfunction to run, the next time at which this subfunction must run must be determined. The "time when run" entry shall be updated to this next time and the time-ordered linked list is reordered with the new value of the time when run entry.

Pointer to Output Message Packet: This word is a pointer to a message packet which contains the initiation information for this subfunction. The packet constitutes the "go" message that is sent by the GEX to an eligible time-dependent subfunction.

Pointer to Forward Element: This word points to the next highest (in time) element in the time-ordered linked list.

Table 32 summarizes the functional description for each entry in the time-ordered list.

The subfunction preamble table and time-ordered linked list are the mechanism used to provide a scheme of centralized, tightly coupled system control at a global level. The time-ordered link list provides a means to ensure periodic subfunctions are scheduled from a common central timing source. Interdependency among such subfunctions is ensured by controlling the relative times (to each other) at which these subfunctions are scheduled from the central source.

### c. *GEX Hardware Usage*

The Programmable Interval Timer (PIT), which has been described in the discussion of the LEX as a potential error-detecting mechanism to ensure that application programs are completed on time, is used in the GEX to derive the timing information on which the GEX scheduling

algorithm works. A software clock is maintained in conjunction with the GEX timer. Whenever a PIT interrupt occurs indicating that it is time for a subfunction to run, this software clock is updated to show the most recent time in the future at which the next subfunction will be scheduled by the GEX. Since the time relationship among subfunctions and their periodicity is predetermined for an avionics environment, a map can be constructed and input as a data base of a time-ordered linked lists to the GEX at system initialization. This map shows the relative time of start of each subfunction and its periodicity. The GEX scheduling algorithm continually updates this map during system operation. It in effect simulates synchronized hardware in that it always tries to schedule all subfunctions that appear in the time map. Overhead is greatly reduced by sending "go" or schedule messages only to subfunctions that are active during a mission phase.

When the system is initialized (see Subsection VIII.J), the GEX scheduling algorithm is given control by the initialization program. As stated previously, it tries to schedule all the time-dependent subfunctions that have been defined in the time-ordered linked list, but sends go messages only to those that have been set active by the process constructor at system definition time. For example, the pilot console application program could be among those set active at initialization. This program could in turn set up other active subfunctions and the Global Executive will then send out more "go" messages to activate subfunctions as instructed by the pilot. Alternatively, subfunctions can in turn activate other subfunctions. The succeeding paragraphs describe each GEX module in detail.

**TABLE 32. GEX TIME-ORDERED LINKED LIST ENTRY DESCRIPTION**

| Table Entry | Descriptor |
|---|---|
| Clock update task flag | This flag if set indicates that this list is for the clock update task which is run periodically to reset the software clock, and time when run for each subfunction |
| Mode flag | Indicates that this subfunction requires mode type information |
| Completed flag | Indicates that this subfunction completed previously |
| Activated flag | Indicates that this subfunction is active and eligible for scheduling |
| Preamble pointer | Pointer to preamble for this subfunction |
| Delta time | Iteration period for the subfunction |
| Time when run | Time at which this subfunction must run |
| Pointer to output message packet | Message packet which must be output to initiate this subfunction |
| Pointer to forward element | Points to next list entry in time-ordered linked list |

The GEX is composed of the following modules:

Scheduler

Clock Update task

LEX functions associated with the GEX PE

Message Output Module

Activate/Deactivate Monitor

Completion Status Monitor

Mode Change Detector Module

## 2. Scheduler (GEXSCHED)

The GEX Scheduler is invoked by an interrupt from the Programmable Interval Timer. The GEX scheduling algorithm is dependent on time-ordered linked list which has been described in a previous paragraph. The flow graph diagram of the Scheduler is shown in Figure 149. As soon as GEXSCHED is entered, the PIT is loaded with a large value so the time spent in the GEXSCHED can be measured.

The Scheduler fetches the pointer to the first element of the time-ordered list of subfunction activation times. It then examines the first word of this initial element. If the word is non-zero, the subfunction which must be scheduled is the clock update task and control is transferred to this module of the GEX. If the first word is zero, the program checks whether the subfunction that must be scheduled had completed successfully the last time it ran, or if the subfunction has been aborted for some reason. In the former case, the program branches to an error routine and in the latter case, the program bypasses several other checks and proceeds to update the time-ordered list.

The program next examines the word which specifies whether the subfunction is active. If the subfunction is inactive, the program again bypasses several other checks and proceeds to update the time-ordered list. Otherwise, the program obtains a pointer to the subfunction preamble pointer from the subfunction's time-ordered element, and determines if all the subfunction's other predecessor conditions have been satisfied. If all their conditions have been satisfied, GEXSCHED requests that a "go" message be output over the Global bus to the PE in which this initial task of the subfunction resides. If the subfunction had any input messages, GEXSCHED requests the LEX scheduler service module. If the subfunction is being scheduled for the first time and has dependent subfunctions, GEXSCHED activates these dependent subfunctions also. The next GEXSCHED task is to update the run time of the subfunction which was just scheduled and reorder its table within the time-ordered list. After the time-ordered list has been updated, the time to run is obtained from the element of the subfunction which is now at the top of the time-ordered list. The clock is updated to reflect this new value. Thus, the clock contains the time at which the next subfunction shall run. The program then reads the PIT and determines how much real time it has consumed. The net time remaining to the scheduling of the next subfunction is determined and if the result is positive the PIT is loaded with this value. The program now restores the original state of the PE and returns to the program location which the PIT interrupted.

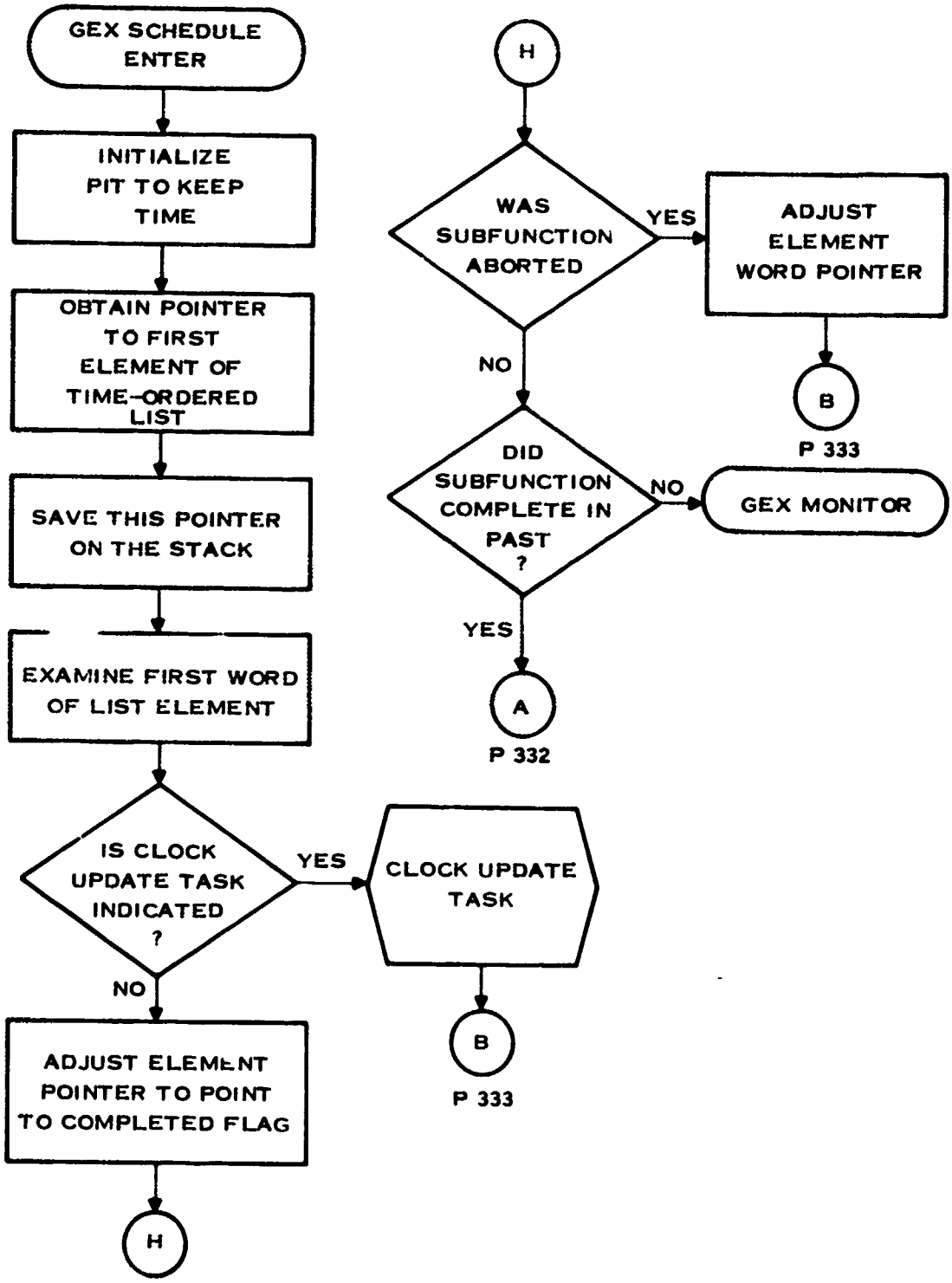



Figure 149. GEX Scheduler Flow Chart (Sheet 1 of 5)

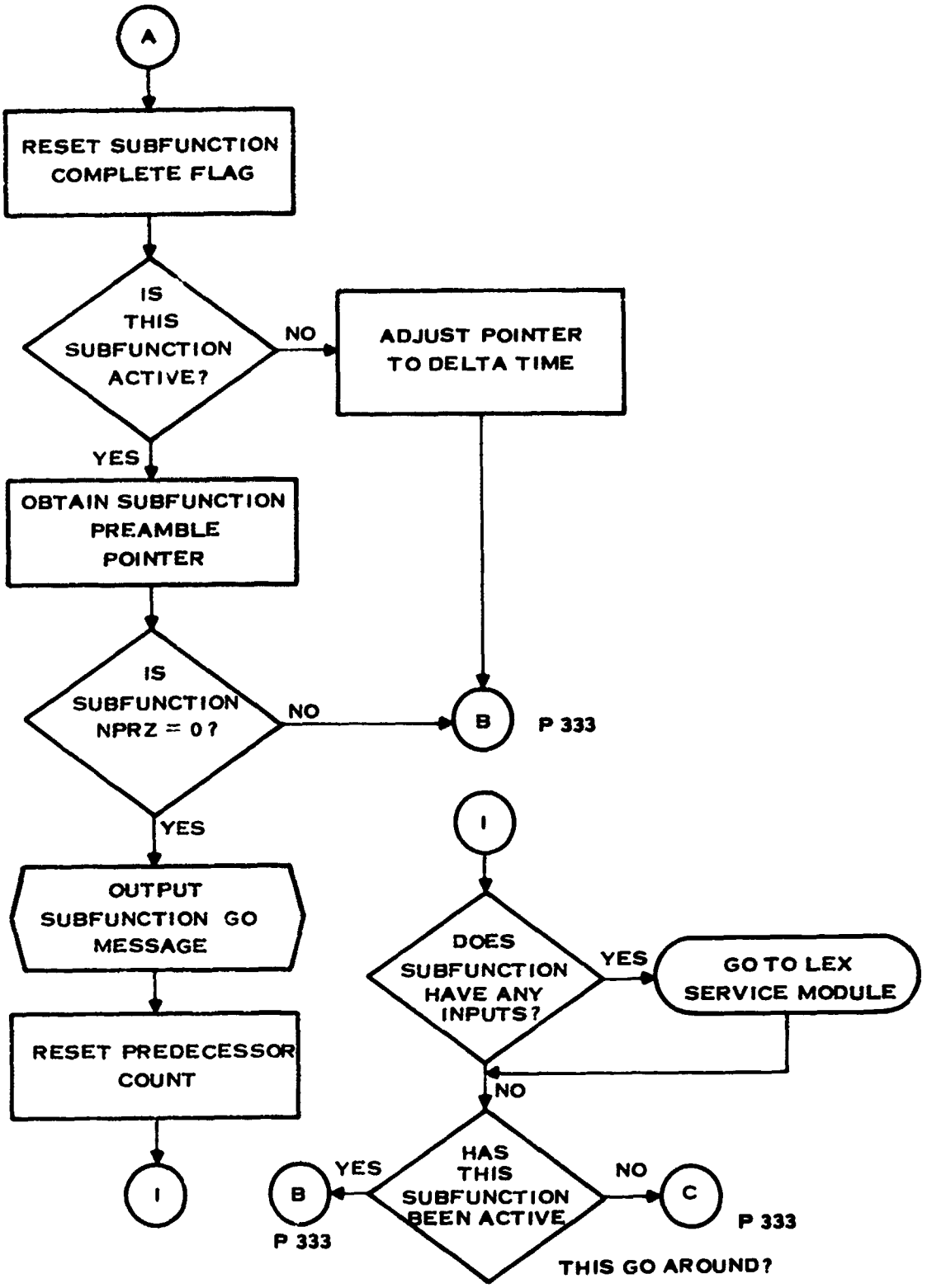



Figure 149. GEX Scheduler Flow Chart (Sheet 2 of 5)

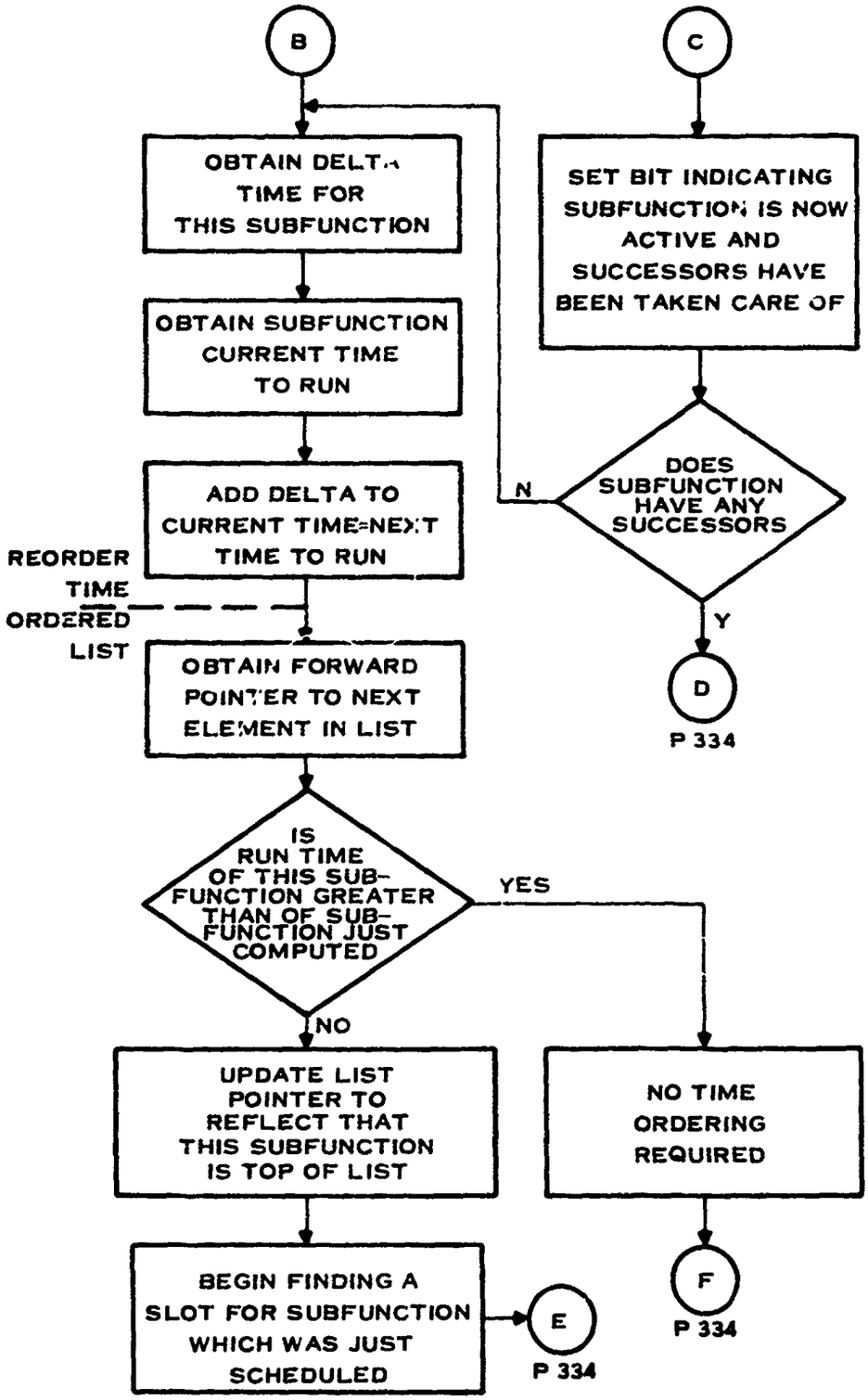



Figure 149. GEX Scheduler Flow Chart (Sheet 3 of 5)

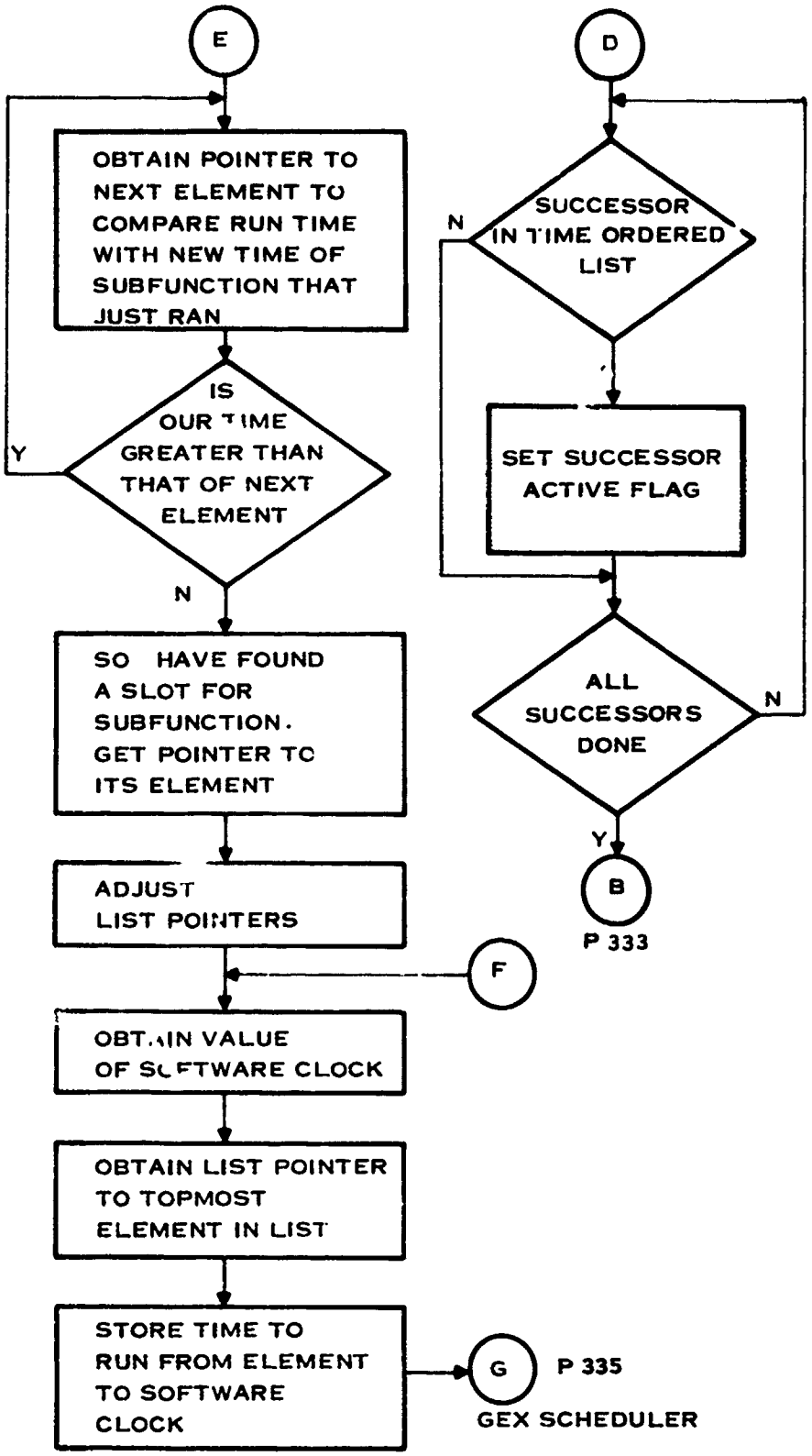



Figure 149. GEX Scheduler Flow Chart (Sheet 4 of 5)

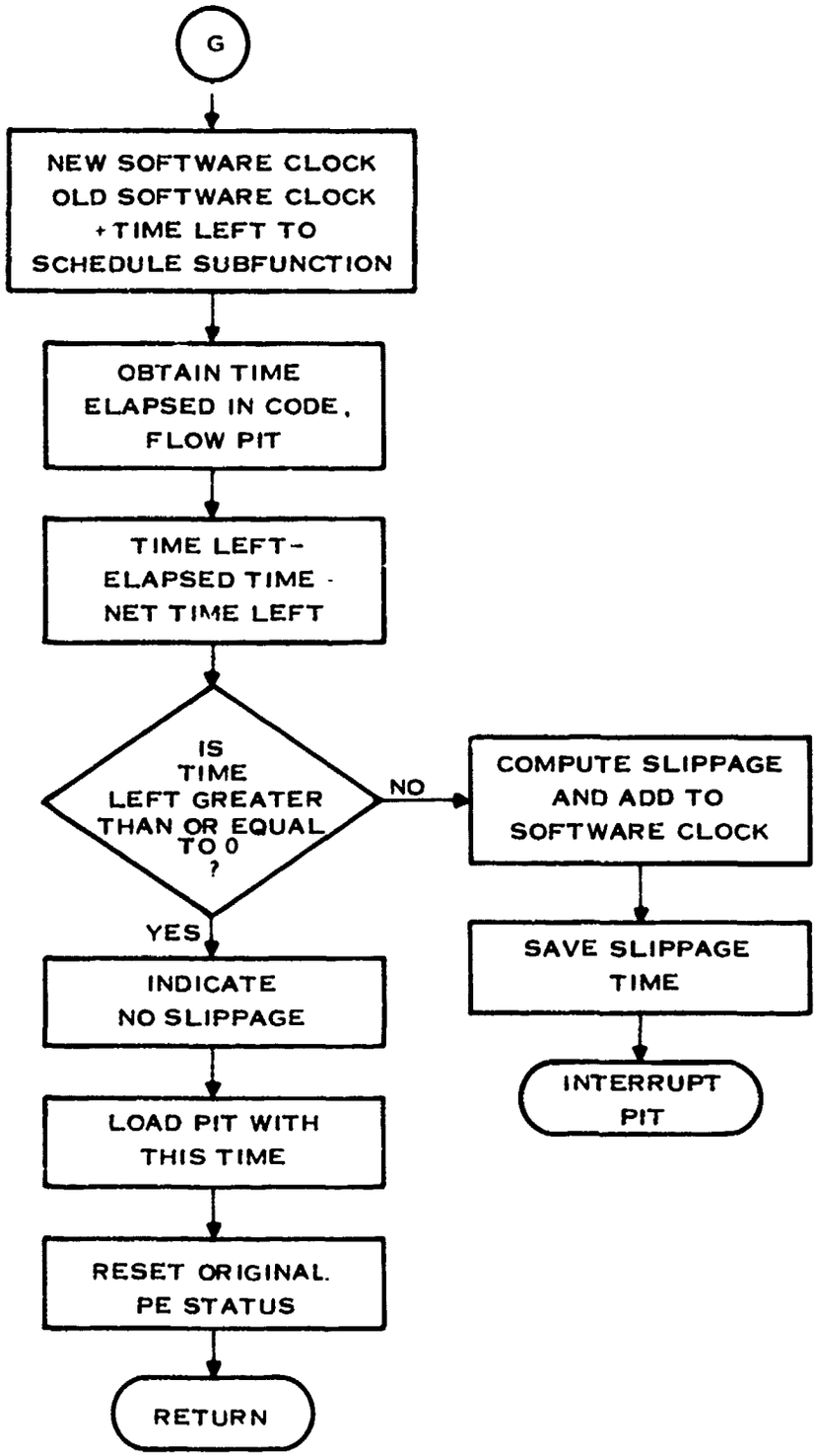



Figure 149. GEX Scheduler Flow Chart (Sheet 5 of 5)

If the scheduler determines that the time to schedule a subfunction has passed, it notes the "slippage" and generates an interrupt by forcing the PIT to a zero value immediately (causing an interrupt) so that a "go" message can be generated as soon as possible. The program also supplies the "slippage" information to the subfunction with the "go" message. In this case, the GEXSCHED shall go through all the processes described above, until it can restore the original state of the PE, and then return to the program location which was interrupted as described above.

### 3. Clock Update Task

The Clock Update Task is time-scheduled by the GEX at a given period. To prevent an overflow from occurring, the clock update task is responsible for devaluing the time-to-run software clocks associated with each one of the time-ordered list elements. The count of the total number of time-ordered list elements is obtained, and a constant value is subtracted from each one of the run time software clocks associated with each list element. The software clock which is maintained by the scheduler is also devalued in the same manner. When these duties are completed, the Clock Update Task returns control to the scheduler, and the scheduler proceeds with updating the time-ordered list.

### 4. LEX Routines in GEX

The LEX in the Global Executive PE performs all functions that have been assigned to the LEX in the DP/M system. In particular, the LEX is responsible for scheduling the two GEX modules: Activate/Deactivate (subfunction) Monitor and the Completion Status Monitor. Messages coming in to the Global Executive, and for the activate/deactivate and completion status tasks are recognized by the LEX Bus Interrupt Service module. The LEX establishes the correlation between these incoming messages and the recipient GEX modules and, should their predecessor conditions be satisfied, the LEX schedules these GEX modules in the same way as application tasks.

### 5. Message Output Module

The Message Output Module in the GEX is responsible for transmitting messages over the Global bus. The module is similar to the Output Message Interpreter Module of the LEX. However, since most GEX messages are transmitted over the global communication bus, it is more efficient to define and implement a special module by eliminating unnecessary local bus interface program code.

Any GEX module which requires message output service calls the Message Output Module with the pointer to the message which must be output. The Message Output Module checks to see if the Global bus interface is busy. If it is, it posts the pointer to the output message into a software FIFO queue which is examined by the Message Transmitter when an output completes its bus transfer. If the interface is idle, the Message Output Module initiates an autonomous transfer over the Global bus, and returns control to the calling GEX module.

### 6. Activate/Deactivate Monitor

The Activate/Deactivate (subfunction) Monitor is scheduled (put into the dispatcher queue) by the LEX, when an activate/deactivate subfunction message is received over the Global bus from a system decision-making entity such as the pilot or a master subfunction. From the

data base available to it, the LEX determines that the recipient of the activate/deactivate message is an activate/deactivate module (task). The LEX decrements the modules predecessor count until it goes to zero and thus, the LEX can schedule this "task."

When the Activate/Deactivate Monitor is given control by the LEX, the pointer to the input buffer list is provided by the LEX to the "task." The first three data words of the input-message contains the activate subfunction information and the next three words contain deactivate information. Each bit in these words corresponds to a system subfunction. A bit that is set implies the subfunction is to be activated/deactivated, while a bit that is a zero represents a noneffected function. The task examines the first data word from the buffer, and if the task determines that the first data word is zero, it then examines the next word. If the data word is not zero, the task examines each bit in the data word and, hence, determines which subfunction needs to be activated.

When it has made the determination of which subfunctions are to be activated, it obtains a pointer to this subfunction's preamble from a table. It then uses the preamble to obtain a pointer to the subfunction's scheduler time-ordered list. The program then sets the active flag in the corresponding element in the time ordered list. When this GEX module has serviced all three "activate" words, it begins servicing the three "deactivate" words in the same manner. It examines each bit, associates a subfunction with each deactivate bit set in the data word, and sets the deactivate flag in the subfunction element in the time-ordered list. When it has completed its processing, the Activate/Deactivate Monitor returns control to the TCEP of the LEX scheduler.

This capability to activate/deactivate sets of subfunctions with one Executive routine provides a method of controlling groups of subfunctions. Note this activation/deactivation provides an orderly way of altering the system state since only the GEX scheduler is allowed to initiate subfunctions. All subfunctions not specifically addressed by the activate/deactivate command remain in their present state (i.e., activate functions remain active and inactive functions remain dormant).

### 7. Completion Status Monitor

The Completion Status Monitor module of the GEX is scheduled (put into the dispatcher queue) by the LEX, when a subfunnction complete message is received in the Global Executive PE. When the LEX recognizes the input message, it correlates it to the completion status monitor "task" which it then schedules. The incoming message contains a pointer to an entry in the completing subfunction's preamble table. When the LEX transfers control to the Completion Status Monitor module, it supplies a pointer to the input message buffer pointer list. The Completion Status Monitor uses this information to obtain the completing subfunctions preamble pointer in the GEX PE. Using the subfunction preamble pointer, the Completion Status Monitor obtains the subfunction's element in the time-ordered list and sets the completion status bit.

Status Monitor has completed when it returns control to the TCEP of the LEX scheduler in the GEX PE.

### 8. Mode Change Detector

The Mode Change Detector module of the GEX is scheduled (put into the dispatcher queue) by the LEX, when a mode change message is received in the Global Executive PE. When

the LEX recognizes the input message, it correlates it to the Mode Change Detector "task" which it then schedules. The incoming message contains mode type information for the aircraft mission. A mode change is associated with a state change in certain subfunctions in the system. For example, a primary sensing device can be determined to be faulty and the subfunction must change its operating mode to use data from an alternate device. This mode change action is equivalent to altering the flow (i.e., branch successors) of the subfunction lattice without impacting the GEX scheduling pattern. When the LEX transfers control to this module, it passes a pointer to the incoming message buffer pointer list to the module. The Mode Change Detector module scans the subfunctions of the time-ordered list that require mode type information. It then obtains the "go" output message pointer for such elements and inserts the mode type information into the output message. The Mode Change Detector module returns control to the TCEP of the LEX scheduler.

## G. SAMPLE ILLUSTRATION OF LEX, GEX OPERATION

To illustrate the operational relationship between the GEX and LEX, an example is presented based upon a simple configuration of avionics subfunctions. Figure 150 is a directed-graph representation of a hypothetical avionics subfunction to be used in this example. The subfunction will be assigned to each of two PEs in two PE Affinity Groups. The other Affinity Group in this system will control the pilot's console. The system consists of three Affinity Groups, with one, two, and one PEs per Affinity Group, respectively. The Global Executive is resident in one PE, the Local Executive is resident in each of the other three PEs. The avionics subfunction identified by the directed graph in Figure 151 is in AG2. A pilot's console interpretive program is in AG3. The relationship among the various executive modules and tasks is shown in a time line in Figure 152.

At system startup, the GEX tries to time-schedule the function in AG2 and the function in AG3. Only AG3 had its activate message programmed into the GEX scheduler table, so when AG3 is to run, an initiation message is sent to the PE in AG3, and the pilot console interface program is activated. If the pilot wants the function in AG2 to run, he depresses a button. This action is interpreted by the pilot's console program task in AG3. This task requests that it has a message to output by calling the Output Message Interpreter Module of its LEX. This message is then sent over the Global bus and is input by the global bus interface of the GEX PE. The Input Message Scanner submodule of the LEX in the GEX PE finds this message in the GEX PEs input

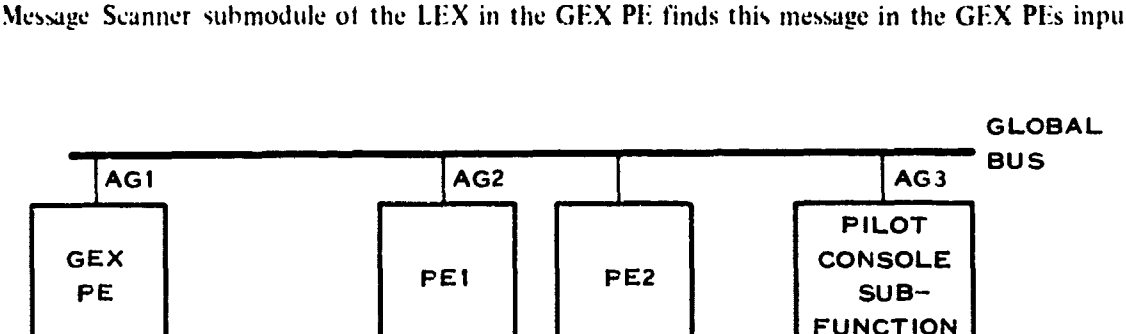



**Figure 150. Hypothetical DP/M System**

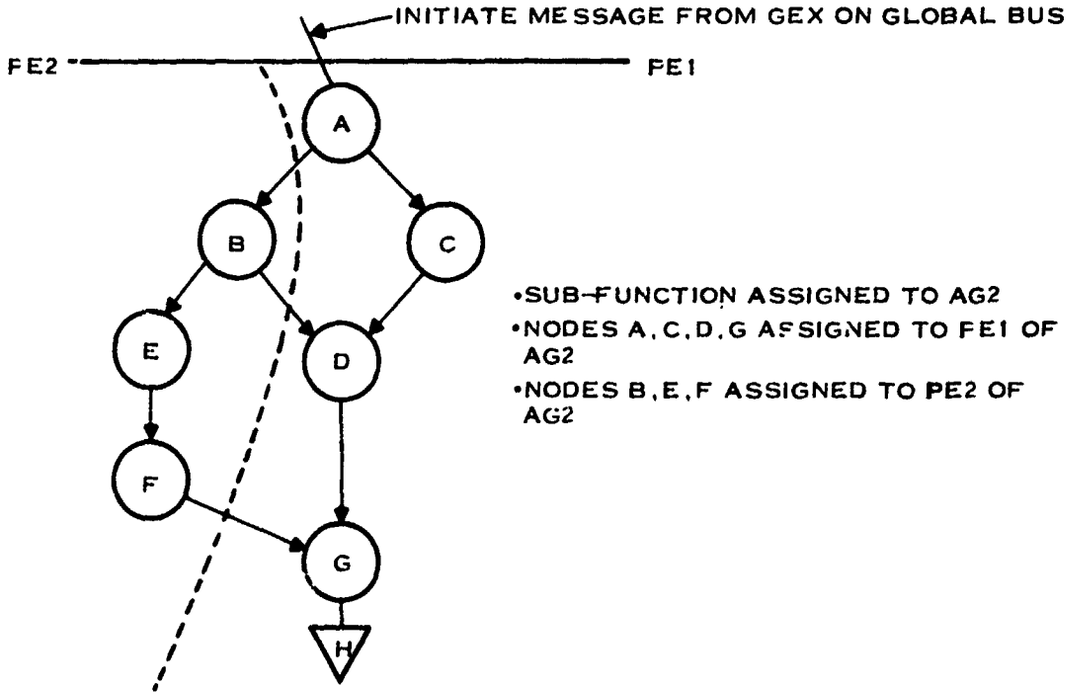
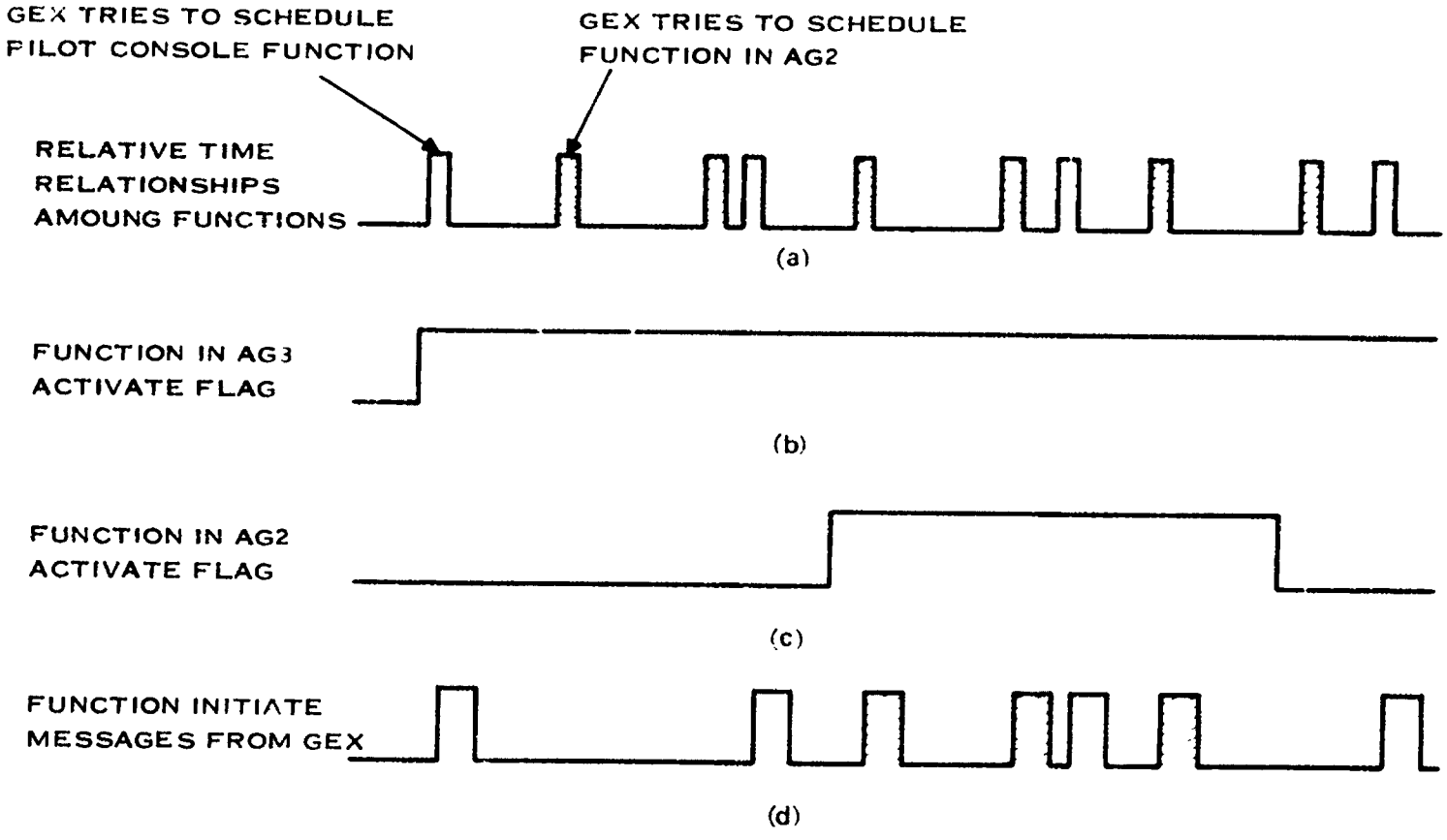



**Figure 151. Hypothetical Directed Graph of Subfunction in Affinity Group 2 (AG2)**

queue, and decrements the predecessor count of the recipient subfunction (i.e., for the subfunction in AG2) in the GEX PE. When it is time to again schedule subfunction AG2, the GEX Scheduler finds the subfunction active and generates an initiate message for the subfunction in AG2 over the Global bus. The interface hardware in PE1 will recognize this message by associative matching, the message ID will be posted into the input queue by hardware, and the message will be routed to an appropriate buffer. When the Input Message Scanner submodule of the scheduler finds that the input message just received is a predecessor for the task A in the task's preamble table, it decrements the predecessor count and moves A to the dispatcher table because its predecessor count is now zero. The Input Message Scanner will look again into the input queue and find it empty, so it will transfer control to the Dispatcher. The Dispatcher will find that task A is at the top-most entry in the task-ready queue and transfer control to it.

When task A runs, it generates two sets of messages: one for task C which is resident and the other for task B in PE2. Suppose A generates a message for B first. Then it will call the Output Message Interpreter Module, which will determine that this message is for a non-co-resident task and is to be sent on the Local bus. It therefore posts an output request in the output message queue and simulates an output message complete interrupt, so the Message Transmitter Module can initiate the message transfer and return to A. The LEX in PE2 recognizes the message, finds that it is predecessor to task B, and decrements B's predecessor count. It finds that it is zero and the process is repeated as described for task A above.

When task A generates a message for task C, it once again calls the Output Message Interpreter Module. The latter determines that the message is for a co-resident task and simply returns control to task A (which has not yet completed). When A completes, it returns to the




NOTES: IN FIGURES (a) AND (d) SHADED PORTION IS SUB-FUNCTION IN AG2. NON-SHADED PORTION IS PILOT CONSOLE FUNCTION IN AG3.

Figure 152. Time Line Description of Illustrative Example

LEX Scheduler at its task completion entry point. The Successor Scanner looks at A's preamble, finds A's successors (viz., C) and decrements C's predecessor count. It finds that it is zero, so it moves C to the Dispatcher table. The input message scanner then looks to see if any input messages have come over the bus. It finds that the bus input queue is zero, so it transfers control to the Dispatcher. The Dispatcher transfers control to task C. Tasks D, E, F, and G are scheduled similarly. Note that node A was the only one that was time-scheduled. Control passed to the other nodes as their predecessor relationships were satisfied.

A special section of code which indicates "end of subfunction" is appended to the last node (viz., G). Thus, when G completes, it will generate this special message. The LEX scheduler will determine from the preamble table that this is a non-co-resident message for the Global bus, and calls the Output Message Interpreter, which then will direct the Message Transmitter to send out this message. The Global Executive will receive this message and examine if the subfunction completed its allocated time. When the subfunction in AG2 is to run again, the GEX will check if it is still active and the cycle will repeat.

## H. SYSTEM OPERATION CONSIDERATIONS/OBSERVATIONS

This subsection discusses certain key areas that were considered in the design of the various DP/M Executive modules. Most all areas that were deemed critical to the satisfactory operation of the system were implemented in the actual design of the executive modules. Other areas were analyzed in terms of their effects on the baseline executive design, and it was determined that many of the probable DP/M processing activities can be accommodated with the baseline executive. Some of the pertinent observations during and after the functional design are discussed in the following paragraphs.

The concept of localized dedicated processing adopted for the DP/M system allowed for much simplification of Local Executive code. The dedication of PEs to subfunctions or of a number of subfunctions of equal scheduling priority eliminates the need for implementing a pre-emptive scheduler in the Local Executive design. Thus, in the normal operating mode, all application programs are allowed to process to completion. Application programs would be interrupted only upon the occurrence of high-priority error detection/recovery events. The dedicated processing concept also allowed that the responsibility of I/O handling be assigned to an application program responsible for processing data from that I/O, rather than to the Executive, thus, I/O processing would proceed on a demand/response basis at an iteration rate determined by the service requirements of that particular sensor/actuator. If the I/O device had interrupt response requirements, the application program would have an I/O interrupt service module incorporated within it to service the I/O. The logic behind this scheme of implementation is that the results generated by the application program servicing such I/O are dependent upon the data input/output to (from) the I/O device. If such interrupts are periodic and predictable, the application program can be scheduled periodically to service the I/O interrupts when they occur. I/O devices with asynchronous interrupts can be serviced in the following manner. The application program which is responsible for servicing this type of I/O would be scheduled by the GEX, at system startup. A module (routine) of this application program would be the I/O interrupt/device-turn-on-monitor module, and would be sufficient to service I/O interrupts from the device. A chain of events could then be based on the device input. For example, such an application task could asynchronously generate output data for other tasks by calling the LEX and, in response, receive input data. The entire operation can be described by a lattice (directed-graph) structure. The PIT which is used by the LEX to monitor an application program's time away from the Executive can be used in this case to periodically

query the status of the applications program which is constantly monitoring the I/O device for which it is cognizant for incoming interrupts.

The other area which required close examination was the processing of messages incoming over the data buses. The area of concern was to maintain the integrity of incoming data in an environment where data could and would be generated at a faster rate than at which it is used. The analysis of this problem suggested the following solution. An application program which uses data at a rate slower than at which it is generated by some other task can use the latest copy of such data, provided the "latest" copy contained a set of data all generated at the same time. In other words, care would have to be taken to give to an application program a set of data which was complete and not being overwritten at the time control was passed to the application program. The logic for implementing this safeguard has been designed into the DP M Executive software and is described in detail in the description of the bus message input modules.

An area that was analyzed but not implemented into the executive code was the capability to re-request data. It was concluded that in an avionics environment where most all subfunctions are periodic, the elapsed time between the receipt of bad data and the next time this data will be generated being based on the iteration period of both sending and recipient tasks, little would be gained by re-requesting data. However, it is a valid contention that the Executive software be able to support re-requesting of data, should this be required. The Executive design is able to support this requirement. A sending task would be required to keep a copy of the data generated by it, until new data is generated. An Executive "task" could then be designed that would be given control by the LEX when the re-request message was received. This Executive "task" could then obtain and put the required data set address into the output queue and preempt processing in the PE to the extent of enabling an asynchronous output transfer of data.

In the previous discussion, it has been stated that the Scheduler module of the Global Executive schedules time-dependent subfunctions in the DP M system when it is time for them to run, and other predecessor conditions (such as the activated status of the subfunction, the past completion of the subfunction, etc.) have been satisfied. It was also stated that tasks are scheduled by the Local Executive only when their predecessor requirements are satisfied. The LEX scheduling function does not have a time dependency, per se. This executive control approach would seem to work well in an avionics environment in which all subfunctions have a relative time-to-run relationship among them. This requires that, when any subfunction is active, it shall be required to run in a fixed time slot relative to the time to run of some other subfunction in the system whether the latter subfunction is active or inactive. The above relationship was the basis for the design of the scheduling algorithm of the GLX. The GEX scheduler tries to schedule a subfunction when it is time for that subfunction to run. Only other predecessor conditions of the subfunction that may not be satisfied at this time (e.g., the subfunction may be inactive) prevent the GLX scheduler from sending a "go" message to the subfunction.

There may be, in an avionics software environment however, certain subfunctions which have no time relationship whatever with any other subfunction. Such subfunctions may become active because of the occurrence of a certain event and become inactive when that event is past. A question might well be asked whether the executive software control system as described previously will be able to handle this type of subfunction(s). Note that LEX scheduling is entirely predecessor-driven. Suppose that a certain subfunction which does not have a need to run is dormant, also suppose that this subfunction does not have a representative element in the GEX scheduler's time-ordered list and, therefore, cannot be time-scheduled from the GLX. Now,

an event occurs in the DP/M system that generates a set of data and the subfunction that has a requirement for this data is the dormant subfunction. If the subfunction is not collocated with the set of data, the PE associated with the data-generating event could use its LEX to transfer the data over the Local or Global communication bus, depending on the connectivity of the sending and receiving PEs to the logically addressed PE containing the dormant subfunction. The receipt of this data would then generate a message complete interrupt; the appropriate LEX module would be called, which would provide the correlation between the incoming data and the dormant subfunction, and decrement the subfunction's predecessor count. Assuming that this count goes to zero, the dormant subfunction becomes active. The task preamble can be structured in the PE so that such subfunctions are self-sustaining. In other words, the subfunction could be made its own predecessor, by merely indicating appropriate relationships in the preamble tables. The preamble tables can also be used in a similar manner to structure "control" tasks around the basic LEX defined in this specification to build a relationship among subfunctions that are related to each other but are not time-dependent. PEs having such subfunctions would still participate in bus control and respond to GEX control messages. Subfunctions could talk to each other and pass messages between each other, the only difference being they receive no time-dependent "go" message from the GEX scheduler. Monitor responsibility for all subfunctions in this subgroup would be decentralized from the GEX, possibly to one of the PEs in the subgroup.

An interesting application of this concept would be the activation of a dormant subfunction in a PE when an infrequent stimulus arrives from the sensor for which it is responsible. As discussed in the above paragraph, a "control" task could be defined in that PE which would be activated when the sensor I/O interrupt occurred. This control "task" would subsequently exit to the LEX and, depending upon the lattice structure defined for the subfunction, a series of activities could be initiated as a result of this action.

The Executive software, as designed for the DP/M system, is therefore quite flexible. By its very nature of being table-driven, it provides separation of system logic and application software modules. Not only can modifications and additions be made to application software modules, but the capabilities of the Executive can be expanded to perform a multitude of functions by defining "control" tasks. This is demonstrated in the design of several modules of the DP/M GEX, and in the suggested nature of various possible Executive software configurations, as discussed in this section.

In conclusion, the executive design is flexible enough to provide for modular addition without requiring extensive baseline design changes. Just as in the case of relative speed/flexibility tradeoffs between micro-programmed and hard-wired computers, a similar analogy can be drawn for the DP/M baseline executive design. The indications from design analysis/simulation have shown that the overhead contributed by the table-driven Executive should be minimal and that the proposed Executive is capable of meeting the real-time requirements of the DP/M system.

## I. SIMULATION MODEL DEVELOPMENT

Part of the DP/M executive design effort was the development of simulation models of various executive routines in FORTRAN and PE assembly languages. Subsection VII.D.12 of the DP/M Functional Simulation section described the event models that were incorporated into the System Network Simulator. These models were used in the Process Construction case study

described in Subsection IX.D. Additional FORTRAN models were developed to check logic used in different executive routines.

The development of certain LEX and GEX assembly language routines was undertaken to determine approximate timing and memory requirements for different executive functions. A summary of these routines and their respective instruction and memory requirements is found in Table 33. Timing equations such as the one described in Figure 153 were formed and used in the execution-time calculations of the FORTRAN System Network Simulator models.

**TABLE 33. ASSEMBLY LANGUAGE MODEL DEVELOPMENT**

| | | Number of Instructions | Number of Memory Words |
|---|---|---|---|
| GEX SCHED | GEX Scheduler | 85 | 101 |
| RCLOCK | Clock Update Task | 15 | 18 |
| OMESS | Output Message Interpretor | 54 | 72 |
| BISVC | Bus Interrupt Service | 11 | 16 |
| MSGXMTG | Bus Message Transmitter | 27 | 37 |
| TCLP | Task Control Entry Point | 11 | 15 |
| BSCODE | Branch Successor Scanner | 8 | 8 |
| GEX DISP | Scheduler Service Module | 88 | 100 |
| SSCODE | Successor Scanner Module | 12 | 12 |
| IMSGCODE | Input Message Scanner Module | 71 | 96 |
| DISPCODE | Dispatcher | 28 | 38 |
| ACTDACT | GEX Activate/Deactivate Monitor | 32 | 40 |
| COMPSTS | LEX Completion Status Monitor | 11 | 13 |

The effort associated with assembly language coding of key executive modules provided a good insight into the adequacy of the DP M instruction set. The extended short format instructions were frequently used and provided good savings in execution times and storage space. The autoincrement instruction and the increment and branch on zero instruction together served as an excellent tool in the efficient scanning of tables whose first word contained the count of the number of entries in the table. Interestingly enough, most of the executive tables have this format and the instructions were put to good use. The direct indexed feature was also very useful in accessing data found at different indexed locations off a common base.

Particularly useful instructions that were identified included the move and autoincrement, the test, set, and clear bit, and the push and pop multiple instructions. The move and autoincrement with the available instruction modifications was used for memory-to-memory data transfers, moving the contents of one table to another, initializing tables, etc. The bit instructions can be used for testing, clearing, and setting flags. These instructions help conserve memory space and execution time, since flags could be compressed in with data words and an elaborate logical mask and set/clear operation would not be required. The push and pop multiple instructions provided a quick efficient means of saving/restoring registers for subroutine and interrupt processing.

$$t_{sched} = 200 + (x_1(12 + x_2 n_1(12 + 8x_3) + 2) + x_4(14 + 15n_2) + 11x_5$$

where

$x_1$ = 0 if subfunction is not running the very first time this segment

$x_2$ = 0 if subfunction has no dependent subfunction

$x_3$ = 0 if dependent subfunction is not in time-ordered list

$x_4$ = 0 if time ordering is not required

$x_5$ = 0 if scheduling has not slipped

$n_1$ = number of dependent subfunctions

$n_2$ = number of moves required to time-order subfunction element

Note: The execution time of the GEX scheduler is dependent upon a number of parameters:

(1) Is subfunction being scheduled for the first time in segment?

(2) Do any dependent subfunctions exist?

(3) Is the dependent subfunction in time-ordered list?

(4) Total number of dependent subfunctions.

(5) Time-ordering required?

(6) How many moves to time order?

(7) Any scheduling slippage?

Further timing information is found in the DP/M Executive functional specification.

**Figure 153. GEX Scheduler Timing Equations**

## J. SYSTEM INITIALIZATION PROCEDURE

System initialization was considered a part of the investigation into types of DP/M system operation. It was important to consider the detailed aspects of system initialization requirements during the system component design to ensure that the initialization procedure is supported and facilitated by the individual functional system hardware design. The following paragraphs describe one suggested method of providing a cold-start initialization for a variety of network configurations, as well as for systems that have variable memory mixes, i.e., ROM or RAM data and program memory, etc. Functionally, the initialization procedure allows the loading of the read/write memory space of all PEs in a given system configuration from a mass memory unit ("program-loader") which has a direct I/O connection to the Global Executive PE. Each PE is provided with a minimal amount of hard-wired control to accomplish the system-initialization procedure.

The DP/M system allows data inputs/outputs locally to each PE over individually dedicated I/O channels, and data transfers among PEs over either the Global or Local buses, depending upon the topological connectivity of the system. The later scheme of data input is implemented by a hardware/software procedure of input message recognition and selection using the Message Identity Associative Match Map (MIAMM) described in Section III of this report. This same scheme of data input is used to load programs into each PE at system initialization. However, since the semiconductor memory of each PE can come up in an indeterminate state at power up, a means must be provided to initialize the MIAMM region of memory to allow proper message recognition/selection procedure in response to system-initialization-control messages. A candidate method is described in the following paragraphs.

### 1. Initial State of the System

When power is applied to the DP/M system, a clear/reset signal provided to each PE by the individual DP/M power supplies initiates hardware-controlled vectoring of the PEs to a ROM containing the PE initialization bootstrap program. The clear/reset signal also brings the entire DP/M system into a known state. In this known state, all programmable (enable/disable) bus interface control registers and interrupts come up disabled. The partial state of the system as pertinent to this discussion is: all PEs have

Global bus interface input data sections enabled

Global bus interface output data sections disabled

Global bus interface bus position and length registers in an indeterminate state

Bus quiescence and dominance detection disabled.

### 2. ROM Bootstrap Program

At power up, each PE is vectored to and begins executing instructions from a ROM bootstrap program. The PE assembly language code for this bootstrap is shown in Figure 154. A bit pattern is inserted as an input message match mechanism into the MIAMM in each PE. This will now enable each PE to recognize and input the software bootstrap program and diagnostics that are subsequently addressed to all PEs. The ROM program in all PEs then

(1) Arms the "input message received" interrupt

(2) Initializes this interrupt's trap location to the RAM address into which the software bootstrap program is to be loaded

(3) Initializes the autonomous I/O data transfer channel (ATC) trap location to the beginning execution address of the program which is to be loaded from mass memory into the GFX PE

(4) Activates the ATC for mass memory input

(5) Finally enters an "idle" state.

Only one PE, the GFX PE which is connected to the mass memory, will receive the system initialization program from the mass memory. When data transfer is complete, the ATC interrupt causes a trap to its reserved memory location. This location, which was previously initialized to the starting address of the start-up program (just loaded from mass memory) vectors the PE to this starting address. The GFX PE is now under program control and system program loading can begin.

```
          LC      R6,STACK          INITIALIZE STACK POINTER
          LC      R3,MIAMM+64       INITIALIZE POINTER TO MIAMM END ADDRESS
          LCS     R1,0              CLEAR COUNTER
          LCS     R2,-64            SET MIAMM TABLE SIZE INDEX
LOOP      PSHP    1,0               CLEAR MIAMM LOCATION
          IBNS    R2,LOOP           ALL CLEARED?
          SBUR,1  R7,R3             YES, SET UNIVERSAL BOOTSTRAP MESSAGE MATCH BIT
          LC      R1,BUFVSP         SET BUFFER VECTOR SPACE BASE
          LC      R3,BOOTSTRT       START ADDRESS OF BOOTSTRAP PROGRAM
          STR,1   R3,R1             PUT START ADDRESS IN BUFFER VECTOR SPACE
          LCS     R4,PROGSIZE       SET BOOTSTRAP PROGRAM SIZE
          STR,1   R4,R3             POSTED AT 1ST WORD OF INPUT BUFFER AREA
          ST,D    R3,ATCTRAP        PRIME ATCTRAP LOCATION VECTOR WITH BOOTADDRESS
          ST,D    R1,GBTRAP         PRIME GLOBAL BUS INPUT MSG COMPLETE VECTOR WITH
                                        BOOTADDRESS
          MCC     LDGSTAT,GSTAT     ARM GB MSG RECEIVED INTERRUPT
          MCC     ACTATC,ATCCAW     ACTIVATE PGM LOAD DEVICE INPUT TRANSFER
WAIT      BS      WAIT              WAIT FOR PROGRAM LOAD COMPLETITION
GSTAT     DC      2
```

**Figure 154. Hardware Bootstrap Procedure**

### 3. Software Bootstrap Program Load

The first record read by the GEX PE from mass memory contains the software bootstrap program (SBP). The GEX PE broadcasts the SBP to all PEs. The input message completion interrupt vectors the PEs to the start of the SBP (the PE assembly language code for the SBP is shown in Figure 155). The SBP executes an I/O activity which reads a hardware-assigned identity (ID) that has been uniquely assigned to each PE. This ID is translated into a bit pattern and inserted into the MIAMM, thereby allowing the PE to identify and load-in system programs that are addressed uniquely to it. If the ROM/RAM mixes of the various PEs in a given configuration are different, this software bootstrap program will be unique to each PE and shall contain the memory address into which system programs for each PE must be loaded. The SBP initializes the MIAMM and the Memory Buffer Vector Space for system program loads into each PE, so that proper routing of incoming messages is allowed. When the SBP has completed, it idles, awaiting transmission of actual system applications programs for the GEX PE.

### 4. System Program Loads

The next program that is loaded from mass memory into each PE is a PE diagnostic program. The GEX PE keeps a copy of the program in its memory and broadcasts a copy to all member PEs of the DP/M configuration. Each PE then executes the diagnostic program and builds a status map. This status map will be output to the GEX when the GEX PE interrogates individual PEs at the time of loading that PE's program. After the GEX finishes its diagnostic test, it executes a timeout to give sufficient time to the other PEs to complete their test. The

```
*                R3 CONTAINS MIAMM ADDRESS
*          R1=0
*          R2=0
SOFTBOOT   RICC   R2,PEID          READ PE  PHYSICAL ID
           LR     R3,R2            SAVE PE ID
           DVC    R1,16            DIVIDE PE ID BY 16
*                                  QUOTIENT IN R1
*                                  REMAINDER IN R2
           AR     R1,R0            POSITION TO PE ID MATCH MAP LOCATION
           OR.I   R2,R1            GET MATCH PATTERN READY FOR MAP
           STP.I  R2,R1            PUT PATTERN INTO MAP
*                                  PE WILL NOW RECOGNIZE SPECIFIC MESSAGES
*                                  ROUTED TO IT
           AR     R3,R2            OBTAIN OFFSET IN BUFFER VECTOR SPACE WHERE
*                                  INPUT BUFFER POINTER RESIDES
           AC     R3,BUFVSP        GET LOCATION
           LC     R5,PGMLOLOC      GET PROGRAM LOAD LOCATION
           STR.I  R5,R3            PUT INTO BUFFER VECTOR SPACE
           L.D    R4,PGMSTRT       GET PROGRAM START ADDRESS
           ST.D   R4,GBT+AP            INTO INPUT MESSAGE COMPLETE INTERRUPT LOC
           LC     R4,07FFF         SET INPUT WORD COUNT
           STR.I  R4,R5            INPUT COUNT
                  .
                  .
                  .
WAIT       BS     WAIT             WAIT FOR INPUT COMPLETE
```

**Figure 155. Software Bootstrap Procedure Code**

GEX then requests the first program from the mass memory. This program has a message header so that it is appropriately recognized by the cognizant PE. When the program transmission is complete, the GEX PE waits for the PE that was just loaded to return status. This status includes the results of the diagnostic test that was previously run and whether or not the program was completed by the PE without any error. After the PE has sent its status to the GEX, it disables it output section, loads its bus position and length registers to appropriate values for subsequent system synchronization (as directed in previous bus message communications), and goes idle. The GEX now requests the next program from the mass memory and routes it appropriately. This same procedure is repeated until all PEs in the DP/M system have been loaded with their respective applications program(s) and data base(s).

## 5. System Synchronization

After the programs have been successfully loaded into each PE, a "system synchronization pending" message is broadcast to all PEs. This message signals all PEs to enable their bus activity watch-dog timers and their output interfaces. The state of the bus position and length registers will have been previously set up by software, such that the GEX will output the first sync pulse (message sync signal) on the Global bus. As this signal propagates to all PEs, the system will be pulled into synchronization as each successive PE's bus position counter gets decremented to zero and reset to the bus length. A similar scheme synchronizes all Local buses in the system with one PE in each Affinity Group given the responsibility of making Local bus synchronization. At this stage, the system is in a fully initialized state and is ready to begin actual mission processing.

## K. SUMMARY OF DP/M EXECUTIVE STRUCTURE

Reiterating statements made earlier, the goal of the DP/M Executive structure design was to investigate the requirements necessary for establishing supervisory control and data transfers between avionic subfunction tasks in a network of PIs. The directed-graph predecessor-successor relationships established a convenient way of representing application tasks for processing by executive routines. The use of sets of predecessor conditions (e.g., inter-PI bus messages) form the foundation for all scheduling activities. The incorporation of this scheduling data into a common table structure resulted in the specification of a simple yet adequate control scheme that is adaptable to multiple-PI configurations. The modules identified for the LLX provide the necessary control and message processing for tasks within a PI. The LLX design is independent of overall system control schemes and can support either the tightly coupled or loosely coupled philosophy of operation. The use of predecessor-successor conditions can be used for directed graphs of functions, subfunctions, and task relationships. This convention offers a convenient means of representing mode changes, error responses, task scheduling, and subfunction scheduling in an avionics environment. The GLX design used these basic features offered by the LLX routines and also provided a basic set of total system control functions. These basic system operations included time scheduling of subfunctions, activating and deactivating subfunctions, and multiple initiation of subfunctions based upon a mission mode change. The detailed design of these total system control functions is dependent upon an exact set of subfunctions, sensors, and a detailed specification of system operating modes. The design specification of these global functions is based upon likely types of operation and is expected to evolve as DP/M is applied to specific mission-processing scenarios.

# SECTION IX
# PROCESS CONSTRUCTION STUDY

## A. INTRODUCTION

This section investigates the problems associated with the construction of real-time processes for DP/M networks. The main objective of this work is to formulate the requirements for a DP/M process constructor and to outline its high-level design, both of which are presented in Subsection IX.C, Summary of Requirements. The formulation of requirements and the development of an approach for DP/M process construction required investigation of secondary problems and finding answers to numerous questions related to process construction. For example, the managerial aspects of the development of process construction capabilities had to be examined and various optimization algorithms had to be investigated. A discussion of these problems appears in Subsection IX.B, Problem Analysis. In addition, an actual process construction exercise was carried out, during which a process model for a partial "Close Support Mission" was constructed by means of computerized tools developed in the course of the work reported here. The objectives of this exercise were not only to demonstrate the current Texas Instruments approach to and capabilities in DP/M process construction but also to acquire a better intuitive understanding of the related problems. The results of this exercise are described in Subsection IX.D, Process Construction Case Study. Possible improvements in the process construction procedure and a suggested future course of action are summarized in Subsection IX.E, Recommended Future Activity in DP/M Process Construction.

### 1. Process Construction Problem Statement

The development of computer-based real-time systems is a challenging task. First, the system to be created must satisfy the timing constraints stated in the design requirements. Its programs must interplay in a temporal order such that the logical system constraints remain satisfied. Furthermore, it must accomplish its mission well, such as controlling a plant, performing on-line services for some external process, or controlling a set of sensors and performing avionic functions. Finally, the system must be reliable and dependable, because the cost of failure may be catastrophic.

Difficulties in the development of software for a real-time computer system typically start to abound in that phase of work during which individual modules must be combined into a working system and when testing of the timing and logical constraints begins. This phase is usually called system integration. Concurrencies in a typical real-time system contribute to the conceptual difficulties that its designer faces and make the system integration job more difficult. The situation becomes even more critical if the system structure is high parallel, as in the case of DP/M computer networks.

During the last few years, great progress has been made in software engineering, structured programming, top-down software development concepts, program validation theory and techniques, and modeling and simulation techniques for software design purposes; the currently active research in programming languages (as well as the appearance of languages such as PASCAL) is a noteworthy recent development. However, most of these developments have overlooked that phase of real-time software development which is concerned with the construction of a framework for software to run in real time and to satisfy all its constraints.

Preceding page blank

Generation of such a framework is known among the practitioners of real-time computer systems as "process construction." (In the present context, the term "process" means the real-time software, including the applications as well as system programs, and all the supporting data base needed to make the software run properly in real time.) There is no universally accepted recipe stating what activities the process construction should precisely cover; this differs from one case to another. Generation of the program loading and real-time control data is often considered as a minimum, but many other aspects of software development and optimization activities may be viewed as being process construction activities. For example, optimal structures for real-time programs may be generated in order to minimize the bus traffic between the processors of a multi-computer system, or the memory paging in real-time, or the amount (cost) of the required hardware, etc. Furthermore, a certain amount of process validation can be done at process construction time. If the validation of assertions about the software correctness (or the process of making the software reliable) is viewed as a multi-stage filter which starts at the time the program is written by the programmer and continues through the actual field tests of the delivered program, then process construction may constitute one stage of this filter. For example, certain assertions about the correctness of process structure (i.e., whether the structure satisfies the required structural constraints, such as the existence of a single entry point for each program, etc.) can be validated at process construction time.

Another way of looking at process construction is that it tries to match software to hardware or vice versa. Individual components of the real-time software are produced and tested by programmers, process construction tools and procedures are used then to combine these into larger units, to assign computer resources to each unit, and to establish communication links between programs. In nonreal-time computing, the user's interface mechanisms with the system, mainly occurring through a job control or terminal language and the linking loader, are the nonreal-time counterpart of process construction. They try to provide the user's program, possibly written in a relatively computer-independent way, with the hardware and software resources needed to execute the program.

During the last few years, various process construction techniques have been successfully used in several real-time software developments. Presently, the construction of processes for conventional computer architectures, such as multi-processor or multi-computer systems, is fairly well understood. As new types of computer architectures, such as microprocessor networks, are introduced and become economically attractive for real-time computing, the process construction functions for them must be newly investigated and established. For example, a microprocessor network architecture, in which microprocessors (each with its own memory) are interconnected through a bus, offers many challenging problems to the designer of the process construction system, mainly because an individual Processing Element (PE) in such a network is likely to have a very limited capacity, but the overall network geometry may be complex. For example, a program in the process may easily exceed the processing capacity of any Processing Element in the network in such a case. Furthermore, one of the functions of process construction is to decompose the violating program into several components, each satisfying the timing constraints and, accordingly, to restructure the process. The way in which the PEs are assigned to process components may affect not only the real-time performance of the system, the complexity of the real-time performance of the system, or the complexity of the executive, but also the amount of needed hardware and therefore, its cost, as several hardware configurations of different complexities may be feasible.

## 2. Overview

The objectives of this section are to:

- Analyze the problems associated with the development of process construction capabilities for a DP M computer network
- Summarize the functional capability and system design requirements for a first iteration DP/M process construction system.

The problem analysis portion of this section (i.e., Subsection IX.B) discusses two aspects of the process construction capability development, the functional aspect and the system aspect. The first is concerned with the transformations and algorithms needed in DP/M process construction. The second aspect deals with the system problems of process constructor development, or with the process constructor as a system which links both the process designer and a computer into a loop. In the analysis of system problems, answers are sought to questions about:

- The scope, or extent, of process construction functional capabilities
- The appropriate level of automation for DP/M process construction
- What is the right approach for developing the process constructor, i.e., whether to develop a full-capability process construction system at once or to build it up gradually in several incremental steps.

The problem analysis portion of Subsection IX.B describes the development methodology on which DP M process construction is to be based. To lay a foundation for discussion of process construction, Subsection IX.B starts with a review of the assumptions about the DP/M hardware and software which are relevant to process construction. The objectives of the functional capability analysis are (1) to characterize the algorithms/procedures that come up in DP M process construction and (2) to identify the key algorithms/procedures and to describe them in detail.

As noted earlier, Subsection IX.C summarizes the functional capability and system design requirements for a first-generation process construction system, called Basic Process Constructor. These requirements are stated in sufficient detail also to serve as a high-level system design specification. The heart of the requirements is the description of the process construction procedure to be used in the Basic Process Constructor. This procedure is to be implemented as a system of computer programs which communicate among themselves through shared data sets. The main inputs to this procedure are (1) the computer-independent design of the process and (2) a model of a DP M Processing Element. The outputs of the procedure define the mapping of the computer-independent process model into a specific hardware configuration. Determination of that configuration is a function of the process construction procedure. Furthermore, this procedure produces (1) process documentation, (2) the control data which is needed to simulate the process or to run it in real time, and (3) the evaluation data addressed to the process designer, which is to inform him about the expected real-time load distribution in a DP/M network. Subsection IX.D describes a process construction use study which was carried out by means of the computerized tools developed in the course of the work described here. The last subsection, IX.E, presents a critical evaluation of process construction investigations and suggests a future course of action.

## B. PROBLEM ANALYSIS

### 1. Basic Concepts and Definitions

The objective of the present subsection is to summarize in one place the concepts and terminology used throughout the discussion of process construction. For that purpose, real-time processes are discussed briefly, then the construction of such processes is defined, and, finally, basic assumptions made about the software and hardware for which real-time processes are to be constructed are summarized.

*Real-time process* is a real-time software data system which consists of the following major components:

(1) Real-time executive(s), including an I/O traffic control subsystem

(2) Applications (i.e., mission-oriented or tactical) programs, which are represented by the software dedicated to perform various avionics functions, such as navigation, flight control, etc.

(3) A start-up subsystem for initializing the real-time operations mission of the entire DPM system

(4) A real-time data base, consisting of

(a) Directories, data set descriptors, and other data whose primary function is to facilitate management of the data classes listed under (b) and (c) below

(b) Executive control tables (to be used by the real-time executives)

(c) Mission support data (to be used by applications programs)

(5) A real-time performance and data collection monitor

Items (3) and (5) listed above are not considered to be vital during the initial phase of process construction research and development; hence, they will be benignly neglected here.

*Process construction* is a nonreal-time procedure for integrating the individual components of a process, identified in the preceding paragraph, into a system which will work in real time and which, in particular, will satisfy the required real-time and hardware constraints. In the case of DPM, process construction includes the following activities:

(1) Analysis of the real-time behavior of individual software modules with regard to

(a) Processor and memory loading, including satisfaction of real-time constraints,

(b) Communication with other software modules and with the outside world.

(2) Allocation of appropriate hardware resources to process components (groups of programs) and the "first-cut" determination of hardware configuration by finding the needed number of DPM Processing Elements (this first determination of hardware configuration does not divide the Processing Elements into Affinity Groups).

(3) Generation of the control framework needed to run the process in real time by producing the control portions of the real-time data base and by linking/loading the programs and the real-time data base.

The term "process decomposition analysis" will be used to refer to the activities encompassed by item (1) above. This term is descriptive, for the main function of the process decomposition analysis is to determine whether a given decomposition of the process into components will meet hardware and real-time constraints. The activities covered by item (2) are best described by the terms "process synthesis" or "process integration," for at this stage constituent process programs are combined into groups and mapped into, or assigned, Processing Elements. The activities belonging to the third group are reminiscent of the conventional program linking and loading. Hence, this stage of process construction often will be referred to as the process linking loading, although actually it covers much more than the conventional linking and loading. For example, various types of control tables for interprogram communication and executive control, normally not considered in the conventional program linking, are generated during the third stage of process construction.

In the investigation of problems associated with the establishment of process construction capabilities for DP M, two problem areas become especially evident. The first one addresses the basic question about what precisely should be done or is needed to construct DP M processes (i.e., what functions algorithms are needed in DP M process construction). The other is concerned with the tools that should be developed to facilitate process construction, as determined by the answers obtained from solving the first problem.

The required process construction tools collectively are referred to as a *process constructor*. One of the objectives of this section is to investigate the requirements for and to specify the design for the initial version of such a system, to be called *Basic Process Constructor*. Briefly, Basic Process Constructor is to be a computerized system of computer programs which intercommunicate among themselves through shared data files, accept inputs from the process designer, inform him of the intermediate results while the process construction procedure is progressing, and, finally, generate the linking loading and the real-time control data. Thus, process construction as viewed here, is a multi-step iterative procedure, including the designer and a computer in the loop.

The extent and the nature of the functions which a process constructor is capable of performing will be referred to as the *functional scope of process construction*. For example, a typical linking-loader, when used through a standard job control language, represents a process constructor whose functional scope is very limited; on the other extreme a highly automated system, incorporating numerous analysis and optimization algorithms, facilitates the development and linking-loading of software systems. Thus, the designer of a process constructor faces the difficult task of determining the appropriate functional scope of the system to be designed. Clearly, the functional scope of a process constructor is determined not only by functional requirements and by technological limitations but also by economies. The problem of selecting a suitable scope for the DP M process constructors is discussed in Subsection IX.B.4.

Another set of decisions which the designer of a process constructor confronts is how much the process constructor should be automated or computerized. It is not only a problem of economics but also that of deciding which tasks can be better or more efficiently performed by man than by a computer, or vice versa. The problem of selecting an appropriate level of automation for DP M process constructors is discussed in Subsection IX.B.3.

The next problem, which to a great extent is of management-decision nature, is to come up with a sensible plan for developing process construction tools and capabilities. A specific approach for developing them for DP M process construction is outlined in Subsection IX.B.4.

This plan is based on, a multistage development of incrementally more powerful process constructors, the first one (already referred to as the Basic Process Constructor) being a subset of all the others.

Before designing a process construction system, a set of basic assumptions on which the development of the real-time process software is to be based must be generated. Software engineers have several options in that respect. For example, they may choose the traditional software development approach based on bottom-to-top integration or they may select the top-down approach which would use structured programming. Either approach may require a different type of process constructor. Consequently, it was necessary to analyze and describe the methodology on which the development of the DP/M real-time software is to be based; this has been done in Subsection IX.B.2.

*a Assumptions About DP/M Hardware and Software*

Next, a few basic assumptions, relevant to process construction, about the DP/M hardware and software shall be reviewed. Since these assumptions are summarized in Section IX.C.1 (i.e., at the beginning of the discussion of the DP/M process construction requirements), they will not be restated here, although they should be read at this point. Here, the discussion will be limited to stating several observations which should complement the summary of assumptions in Section IX.C.1:

(1) One remarkable feature which permeates throughout the design of DP/M is the two-level hierarchy, which manifests itself in hardware and soft ware design. In hardware, the two-level hierarchy occurs through the organization of the microprocessor network into clusters of Processing Elements, called Affinity Groups, and through the Global and Local bus systems. In software, the two-level hierarchy occurs through it organization into *programs* (along the natural boundaries of avionics and other mission-related functions) and of each program into smaller functional segments, called *tasks*. It also occurs in the design of the current real-time operating system, through a two-level control system consisting of a Global and a Local Executive. Consequently, this two-level hierarchical approach also shows itself in process construction. On the lower level, each process program is analyzed as to whether it can be accommodated in a single Processing Element without exceeding the hardware and real-time constraints. On the higher level, programs are grouped into clusters and mapped into individual Processing Elements (or, if viewed from a different angle, Processing Elements are assigned to such program clusters) in a way which leads to minimization of the bus traffic and, thus, the minimization of hardware.

(2) In Subsection IX.C.1, practically no assumptions about the real-time Executive are made, for it is expected that the design of this system will further evolve and change in time. For this reason, the phase of process construction which most closely interfaces with and depends on the Executive design (i.e., generation of control tables and program linking) could be investigated and discussed only in functionally generic terms. For example, it would have made little sense to specify now in great detail various Executive and I/O Control tables and buffers if their designs will almost certainly change. Furthermore, it is presently thought that this last phase of process construction is straight forward in nature because it consists of simple bookkeeping and data processing tasks.

(3) Finally, the choice of the programming language for coding the process software greatly affects the initial phase of process construction, i.e., that phase which is mainly concerned with obtaining and properly organizing all needed information about the process to be constructed. The importance of the programming language comes not only through the language constructors, such as those implementing the process inter-module and external communications or the storage management for variables used by a program, but also through the auxiliary information about the process structure that can be produced as a by-product of compilation. It is widely believed that the ultimate success of microprocessor networks depends on the emergence of suitable programming languages and compilers. However, in the study being described here, only very minimal assumptions about the programming language were postulated. The whole issue of programming languages could be avoided at the present time by choosing a multi-step approach for developing process construction capabilities: the first process construction which is to be produced in this development cycle and which is specified in this report only assumes the availability of a conventional higher level programming language such as FORTRAN. This naturally has been done at the cost of not being able to automate the initial phase of the process construction procedure to the extent it would have been possible had a special higher level programming language and a special compiler been assumed.

As a result of what has been said above in remarks (2) and (3), the attention during the process construction investigations discussed presently was largely focused upon the process analysis-decomposition and the process synthesis and resource allocation problems. The other two problems (interfacing with a special programming language and its compiler and interfacing with the real-time operating system) were identified as being important but, because of the time limitations, were given reduced emphasis.

### *b Process Representations*

One of the first steps in process construction is generation of abstracted process representations which would contain all subsequently needed information and be convenient to work with. This information can roughly be divided into that describing the process structure and that defining the physical attributes of process components.

The first type of information describes various aspects of process dynamics, at least two of which are of interest to the process designer. The first aspect defines the information flow through the process during its execution, the other control dynamics. These two aspects of process dynamics can be represented by two graphs, the data-flow graph and the control graph, respectively. Both graphs are discussed in the sequel.

Physical attributes of process components describe each component such as a program or a data set, in terms of the characteristics that are of interest in a given situation. For example, in high-level network simulation of the process, the physical attributes describing a program may include an estimate of the program execution time, its memory requirements, and its inputs and outputs. The physical attributes of a process normally are represented in the form of various data structures, such as tables, lists, etc.

Next follows a discussion of a general approach for graphically representing the two aspects of process dynamics mentioned earlier, information flow and control dynamics. This

approach is based on what in the theory of operating systems is known as programming (or process) schemata. To simplify the discussion, a one-level process model is considered in the sequel; extensions to a two-level hierarchical process model are self-explanatory.

### (1) Process Schemata: A Generalized Approach for Representing Real-Time Processes

A process schema is a representation in graphical form of an asynchronous process consisting of a set of functional tasks and control boxes. Each task, similarly to mathematical functions, acts on input arguments to produce a set of output values. The process execution logic defined by control boxes specifies the task execution sequences.

The input arguments of a task are located in its input files (data sets); the output values are stored in its output files (data sets). Each task in the process must have at lease one output file and may have none, one, or several input files. Analogously to constant mathematical functions (i.e., those that have no arguments), a constant task is one which uses no input arguments and modifies every record in all its output files in a fixed and predetermined way, independent of the current contents of the output files. To allow representations of iterative tasks, a given file may function both as an input and an output file of an iterative task. This last feature may be used to represent modifications (single- or multiple-step) of the values shared in a data set.

All exogenous inputs into the process and the endogenous outputs produced by the process are represented by means of the external input and the external output files, respectively. Thus, the interface of the process with its environment may be completely specified by defining the inputs (including their arrival rates) which are pumped into the external input files and similarly, the outputs (including their production rates) which are put into external output files.

The control mechanism of the process manages the process execution logic by changing the task states and determines alternative execution paths (task execution sequences). The control mechanism is defined by a network which contains nodes of two types, control boxes and functional tasks (from now on, functional tasks will be briefly called tasks).

In this network, every node representing a task must have a single exit point connecting to a path (outpath) which leads either to the successor task or else to a control box; furthermore, every task must have at least one entry point, each such point corresponding to an incoming path (inpath).

One can divide the control boxes of a process model into three classes according to the number of inpaths and outpaths that they possess:

(1) Entry control boxes: such a box has zero inpaths

(2) Exit of termination control boxes: each of them has zero outpaths

(3) Intermediate control boxes: each of them possesses at least one inpath and at least one outpath.

Every process model must have at least one entry control box and at least one exit control box.

An inpath to a control box may emanate from a task or from another control box; similarly, an outpath may point either to a successor control box or to a task.

At any instant during program execution, the state of a control box is represented by a non-negative integer (which tells how many control outpaths are currently open) the list of open control outpaths, and for each open outpath a specified time delay relative to some absolute time point. The zero state means that all control outpaths are currently blocked, which implies that any process execution path passing through the control box in zero state cannot progress beyond the box until its state attains non-zero value. A non-zero state of a control box indicated by the positive integer, I, implies that I control outpaths are currently open. A companion list tells which control outpaths belong to the set of open outpaths. For implementation, the easiest thing to do is for each box to maintain a list of all its control outpaths, and then to indicate by an outpath status bit whether a given outpath is closed or open.)

The execution logic works as follows. At the initialization of process execution, at least one entry control box is assigned to a non-zero state, all other control boxes are put into the zero state. When a control box is entered during process execution, the following steps are carried out:

(1) The predecessor control box is appropriately modified or reset, which typically amounts to closing the path just used in order to enter the current box. (An entry control box skips this step.)

(2) The state of the current control box is analyzed and, if necessary, changed by closing or opening control outpaths and by computing the delays for each path just opened.

A process schema is completely defined by means of two directed graphs, a data flow graph and a control graph. These two graphs are discussed next.

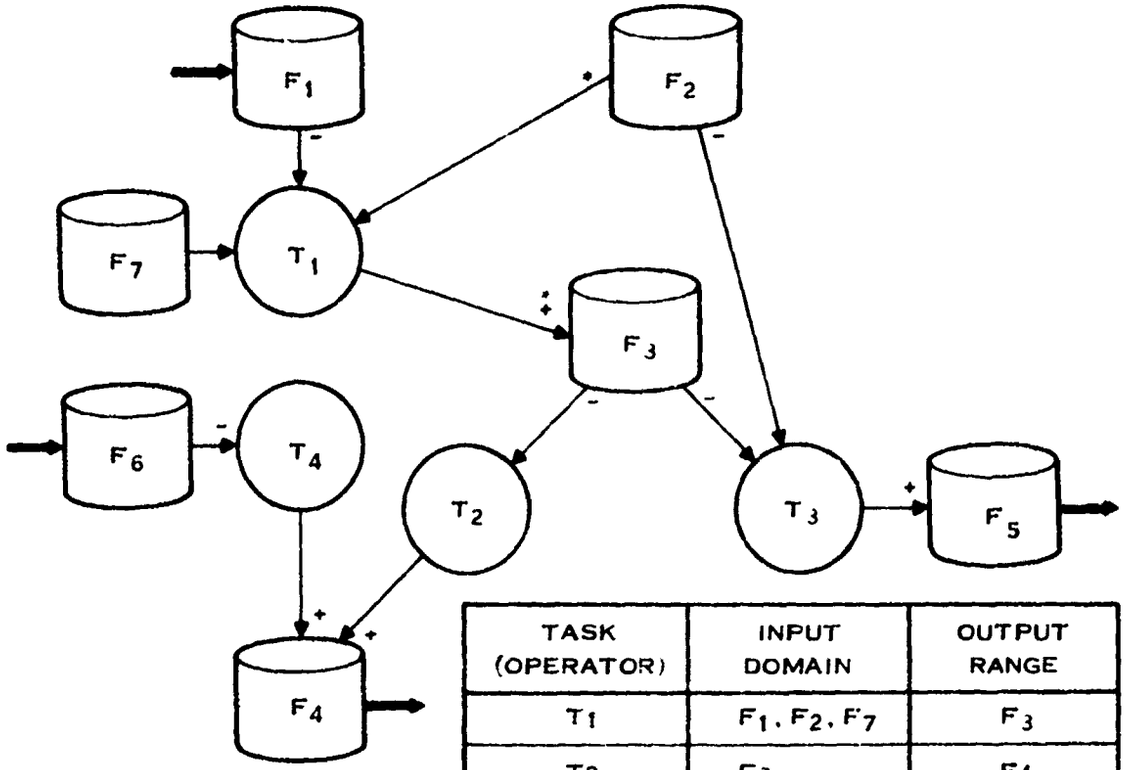
(2) The Data Flow Graph

The data flow graph specifies the input domain and outputs of each task in the process model. This directed graph contains two types of nodes, files (or data sets), and functional tasks (or operators). These nodes are interconnected by directed links (or edges). A directed link from a File, $F_1$, to a task, T, indicates that $F_1$ is in the domain of T; similarly, a directed link from T to a file, $F_2$, shows that $F_2$ is in the range of task T. As already indicated, the domain of a task may be empty or may consist of one or several input files; the range may consist of at least one output file.

The following definitions will be used for discussing data flow graphs. A file (or data set) will be represented by the standard flow charting symbol for files; a task will be represented by a circle.

The following example, Figures 15a and 15b, shows the data flow graph and a companion control graph of a simple process model. The first figure illustrates an important principle according to which a data flow path from one file to another always goes through a task node, but there may be no file in a path between two successive tasks (for example, tasks $T_1$ and $T_2$).

To denote that there may be a net loss of records in an input file as a result of the execution of the task in whose input domain the file is located, the minus sign (-) will be put next to the directed link from the file to the task; if the input file is a read-only, no indicator will appear next to the directed link. A directed link which functions as an outpath from a task




| TASK (OPERATOR) | INPUT DOMAIN | OUTPUT RANGE |
|---|---|---|
| T1 | F1, F2, F7 | F3 |
| T2 | F3 | F4 |
| T3 | F3, F2 | F5 |
| T4 | F6 | F4 |

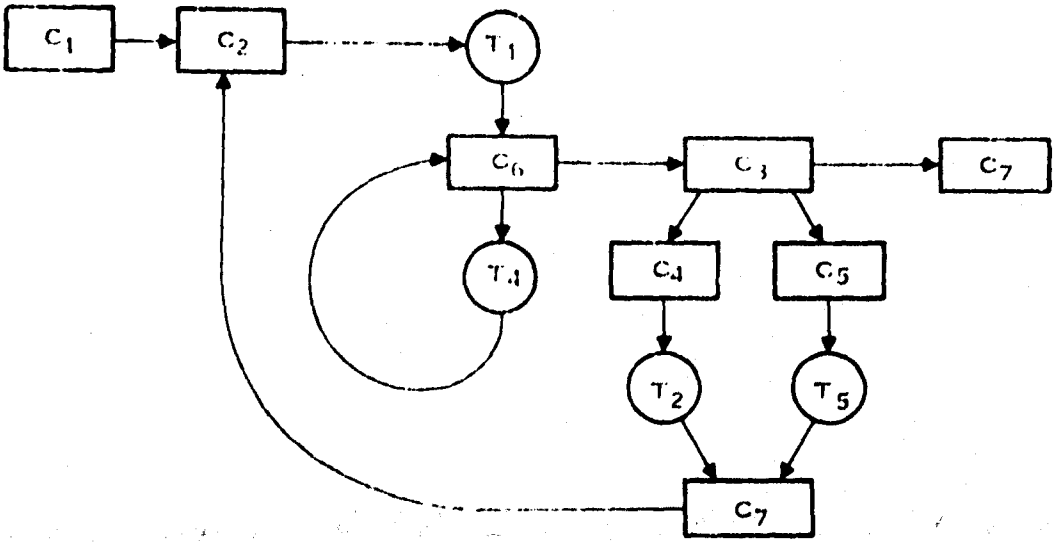
FILES $F_1$ AND $F_6$ ARE EXTERNAL INPUT FILES FILES $F_4$ AND $F_5$ ARE EXTERNAL OUTPUT FILES. FILE $F_7$ IS A READ-ONLY INPUT FILE IN THE DOMAIN OF $T_1$.

**Figure 156. Data Flow Graph**

to an output file will always be associated with at least one of the following two symbols, an asterisk or a plus sign. An asterisk denotes that the contents of the output file may be modified during the execution of the task, the plus sign tells that the execution of the task may cause a net gain in records in the output file. The rules which specify the data flow from an input file to output files as a result of the execution of a task belong to the functional definition of a task.

(3) The Control Graph

This directed graph describes the execution control logic in the process. It consists of nodes of two types, one representing the functional tasks (same as in the data flow graph) and the other represents the control boxes. As discussed earlier, every control box in a process model is of one of the following three types: an entry control box (it has no inpaths); an exit control




CONTROL GRAPH OF A PROCESS WHOSE DATA FLOW GRAPH IS SHOWN IN FIGURE 156. CONTROL BOX $C_1$ IS THE ONLY ENTRY CONTROL BOX IN THE PROCESS. $C_7$ IS THE ONLY EXIT NODE. ALL OTHERS ARE INTERMEDIATE CONTROL BOXES.

**Figure 157. Control Flow Graph**

box (it has no outpaths), or an intermediate control box (it has at least one inpath and at least one outpath). Note that the functional tasks are completely shielded from the system environment in the sense that every possible execution path must start with an entry control box and end with an exit control box. Two kinds of symbols are used to represent the nodes of a control graph: rectangles represent the control boxes; circles, as before, represent functional tasks. A complete specification of process control requires that the status of control boxes be properly initialized and that the algorithms governing the functional behavior of control boxes be specified.

(4) Program Graph and the Maximally Parallel Predecessor-Successor Graph

The approach used to represent process topologies in the design of the process construction procedure described here is more restricted in the sense that it is based on two more specialized graphs, the so-called program graph and the maximally parallel (determinate) predecessor-successor graph of a program. The second of these two graphs is more general than the first in the sense that the first can be derived from the second. Both graphs, however, are derivable from the information contained in the data flow and control graphs.

The program graph is a directed graph which defines all alternative execution paths through (the model of) a program. Its nodes represent the tasks (functional code segments) constituting the program. For process construction purposes, the following assumptions are made about the program structure and thus about its program graph:

(1) There is a single entry node (task)

(2) There is a single exit node (task)

(3) There are no closed loops through the nodes (i.e., all program loops must occur within individual nodes (tasks)).

The maximally parallel predecessor-successor graph of a program contains more information than the program graph. For each alternative path in the program graph, it shows all parallel tasks or strings of such tasks which can be executed concurrently without any loss in process determinancy. The descriptor "maximally parallel" says that any other similar graph showing more parallelism would not guarantee process determinancy (i.e., program output would not be independent of the execution sequence of parallel task groups). Furthermore, this maximally parallel predecessor-successor graph is the simplest possible of all such determinate graphs, as other such graphs would be more complex by having more edges (links) than the simplest one. In that sense, the maximally parallel (determinate) predecessor-successor graph is unique for a given program model.

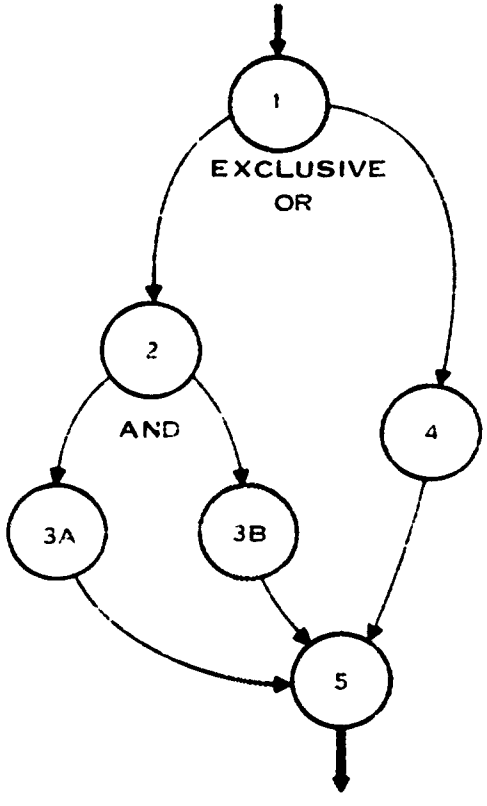
To illustrate the relationship between the maximally parallel predecessor-successor graph of a process and the resulting program graphs, consider the example given in Figure 158. The process model shown in the maximally parallel predecessor-successor graph possesses two mutually exclusive execution paths: the first one passes through nodes 1, 2, 3A and 3B, and 4, the other path goes through nodes 1, 4, and 5. The only parallelism that exists in the model is contained in the first path (nodes 3A and 3B). Using the graph on the left, one can construct, as shown in Figure 158, two program graphs. This example illustrates two points: first, the program graph is not unique; secondly, a program graph contains less information about the process than the maximally parallel predecessor-successor graph from which the former has been derived.

The program graph has been used in process construction mainly to analyze whether a program does not exceed the capacity of a DPM Processing Element, the maximally parallel predecessor-successor graph to construct the task execution sequence tables and the task enablement conditions for the DPM Executive. Both graphs are further discussed in the sequel.

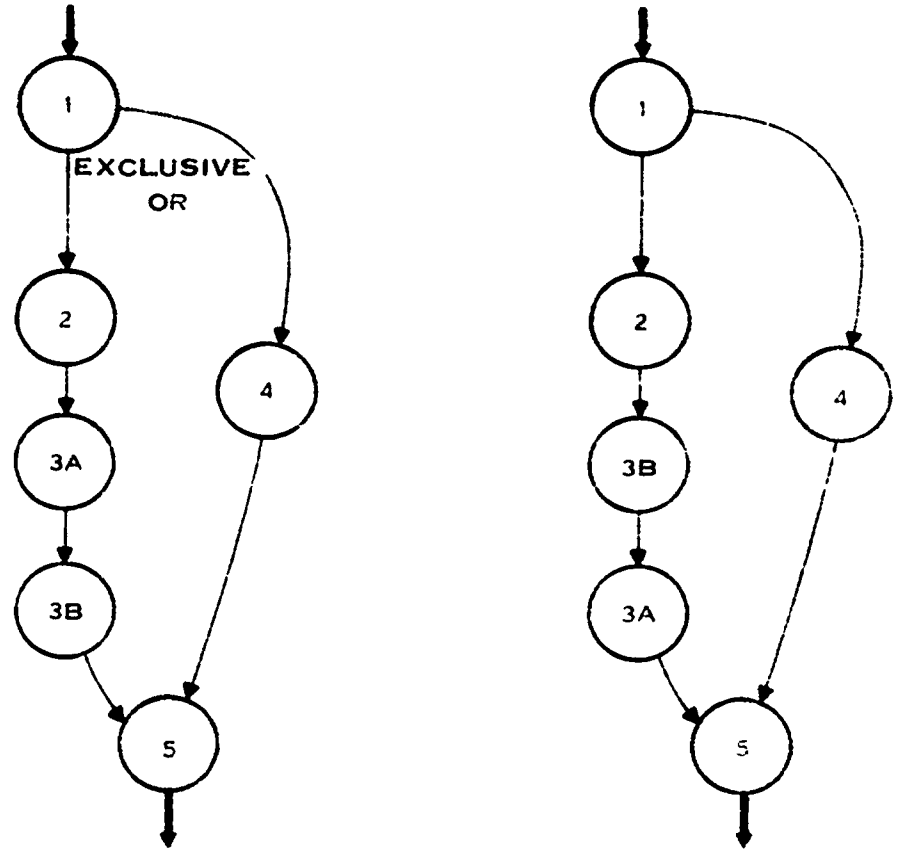
## 2. Process Development Methodology Based on Top-Down Modeling and Successive Simulations

The process construction procedure under consideration is intended to be applied to process development that is primarily based on top-down modeling and on successive simulations of the entire real-time process. This approach still permits occasional bottom-to-top development of individual process components whenever it appears to be practically advantageous; however, at any stage of process development, it must be possible to integrate individually developed components into the overall process model and test them in the simulated environment of the DPM network.

Since the basic process construction procedure interacts with successive simulations of the process in the sense that the process construction outputs are to be simulation inputs, a good understanding of the assumed simulation methodology is essential to understanding of process construction. Therefore, this subsection is dedicated to a review of the process development methodology based on successive simulations.




MAXIMALLY – PARALLEL
PREDECESSOR – SUCCESSOR
GRAPH OF A PROCESS




TWO FEASIBLE PROGRAM GRAPHS

NOTE THE POSSIBLE PARALLELISM IN THE EXECUTION OF TASKS (MODES)
3A AND 3B IMPLIES THAT THEIR EXECUTION ORDER DOES NOT AFFECT
THE DETERMINACY OF THE PROCESS.

**Figure 158. Relationship Between the Maximally-Parallel Program Graphs**

In the course of evolutionary specification, the process under development goes through a metamorphosis in the following steps:

- It starts with the high-level specification of the application real-time system (hardware and software) on a host computer.
- Gradually, high-level models of process components are one-by-one replaced with their more detailed counterparts. This replacement process is not necessarily monotonic, but one is also allowed to go backwards from a more detailed model of a process component to a cruder version of it.
- It finally appears as a package to be loaded and executed on the target computer system or as a DPM support.

It is assumed that process design and implementation will proceed in an evolutionary fashion in such a way that at each stage of development the process software will consist of a mixture of modules. Some of these modules will represent detailed models of process components, the others are yet-to-be-specified ones. This at any time in the course of process development, the simulator to be used for that instant must be able to handle model mixtures. For example, the Executive is modeled functionally in the DPM System Network Simulator, whereas only synthetic models (running times, storage requirements) are used for applications (mission-oriented) programs.

Four basic model types are used to represent process software components in simulations discussed here:

Computer-independent models

Synthetic models

Functional models

Analytic models

Description of these models follows next.

### *a Model Development*

#### (1) Computer-Independent Models

Computer-independent models are used to represent process software components at the beginning of process development, just before starting simulation. This type of model derives its name from the fact that it describes the program execution dynamics in a computer-independent manner, which can be accomplished by specifying (a) counts of various operations (addition, multiplication, etc.), (b) memory requirements, (c) program inputs, outputs, and subroutine calls, and (d) alternative program execution paths. In early stages of process development when detailed information about the logical conditions or rules governing the transition from a node to its successors are lacking, probabilities can be used to define branching through alternative execution paths. Furthermore, it is assumed here that the graph of a structurally correct program has a single entry node, a single exit (which will still allow having variable exit conditions), and will have no cycles. The processing load due to a loop (such as a DO-loop) within the program shall be represented within a single node of the program graph. We do not consider these requirements to be very restrictive because if we go down to the next greater level of detail we can think of any node in the program graph or in the maximally parallel predecessor-successor graph as representing another program.

As already noted, there is a second graph, related to the program graph, which also belongs to the computer-independent model and is used to define all possible parallelism in the execution of a program: the maximally-parallel precedence-successor graph of the program. Note that the topological model of a program obtainable from either one of these program graphs is more general than that allowed for structured programs. The program graph of a structured program is a special case of the graphs described above because it required that all nodes be completely ordered.

The underlying reason for introducing computer-independent models is that such models constitute a convenient medium to study the effects of changing the computer hardware parameters on the same process (system of programs). Past experience has indicated that the most efficient approach is to construct a computer-independent model for the applications (mission) part of the process and then to use the data base thus generated to get computer-dependent, running-time models for various Executive and/or computer configuration combinations as needed. Such a process of transitioning from computer-independent to computer-dependent models can be considerably accelerated by a computerized procedure which reads (1) the computer-independent description of a program and (2) the parameters of the computer under study and then produces a computer-dependent description of the same program. Such a computer-dependent description of a program, to be discussed next, is called its synthetic model.

(2) Synthetic Models

As noted above, the synthetic model of a program is produced by merging two sets of input information, one of which describes the computer under consideration; the other describes the computer-independent model of the program. Thus, a synthetic model always implies a specific computer. Synthetic models are used to represent programs both in system network and in functional simulations. In the latter case, a synthetic model usually constitutes just a part of the functional model. A complete synthetic model of a program contains the following information:

(1) Program execution control data (scheduling rates, priority, etc.)

(2) Its memory requirements stated in terms of the word count for the computer under consideration

(3) Specification of all feasible execution paths through the program graph (including the logical conditions or probability for each path)

(4) Predecessor-successor graph, indicating the partial ordering of and parallelism in program segments (tasks).

(5) Estimated limits on the running time for each feasible path and for the individual tasks constituting the program

(6) Description of the program I/O.

(3) Functional Models

Functional models of tasks and programs are used mainly in functional simulation. For the time being, it will suffice to think of a functional simulator as being an extension of the System Network Simulator in which synthetic models have been replaced with their functional counterparts. The consequence of this replacement is that functional simulation can produce and

consume the value data of the simulated system, whereas system network simulation is restricted to handling dummy (i.e., abstract) representations of this data.

A functional model of a program includes its synthetic model; in addition, it must contain the functional model of the algorithm (or computational procedure) which is to be represented by the program. Although a functional model may be just a crude version of the algorithm (procedure), it still must be able to represent certain salient features of functional behavior that are of interest to the system designer. As noted, one important characteristic of a functional model distinguishing it from a synthetic model, is that it handles the actual value data of the modeled system, in contrast to a synthetic model, which handles only the dummy (abstract) representations of the same data. Furthermore, a functional model may know about the true state of the simulated system and its environment, whereas an analytic model (discussed next) is kept from this knowledge and learns about the state of the system or its environment only through the incomplete or contaminated information that would flow into and through the actual system.

In summary, functional model programs may consume or produce actual value data of the simulated system, may be modeled on many different functional levels, and may have access to the information about the true state of the system and its environment. Such a model always contains as its kernel the synthetic model of the same program.

(4) Analytic Models

The actual implementation of a program or task on the target computer represents its analytic model. It makes relatively little difference whether this model has been coded in the assembly language of the computer under consideration or a high order language has been used for that purpose. (What is important in the latter case is that the appropriate language translator for the target computer exists.) Furthermore, the analytic model does not have to represent the ultimate version of the algorithm or of a computational procedure; it may just express a crude version of the algorithm (procedure). What distinguishes the analytic model from the last two types just discussed is that it possesses no synthetic kernel but describes its own dynamic behavior by being executed on the target machine or on its Instruction Level Simulator.

In the development of real-time processes for DP/M, the strategy should be to introduce analytic models as late as possible in the development cycle; they become unavoidable in software test beds and in predeployment acceptance tests. Otherwise, due to the limitations of a DP/M network to support its own software development, the bulk of developmental simulations should be carried out on a host machine; thus, those simulations will be either of a system network or of a functional type. One separate type of application where analytic models may be used is instruction-level simulation of individual software modules to validate synthetic models for system network and functional simulations.

Figure 4-9 summarizes the evolution of models and shows their temporal ordering during system development.

*b. Simulation Modes and Simulators*

The following basic simulation modes are associated with the proposed methodology for system development:



Best Available Copy

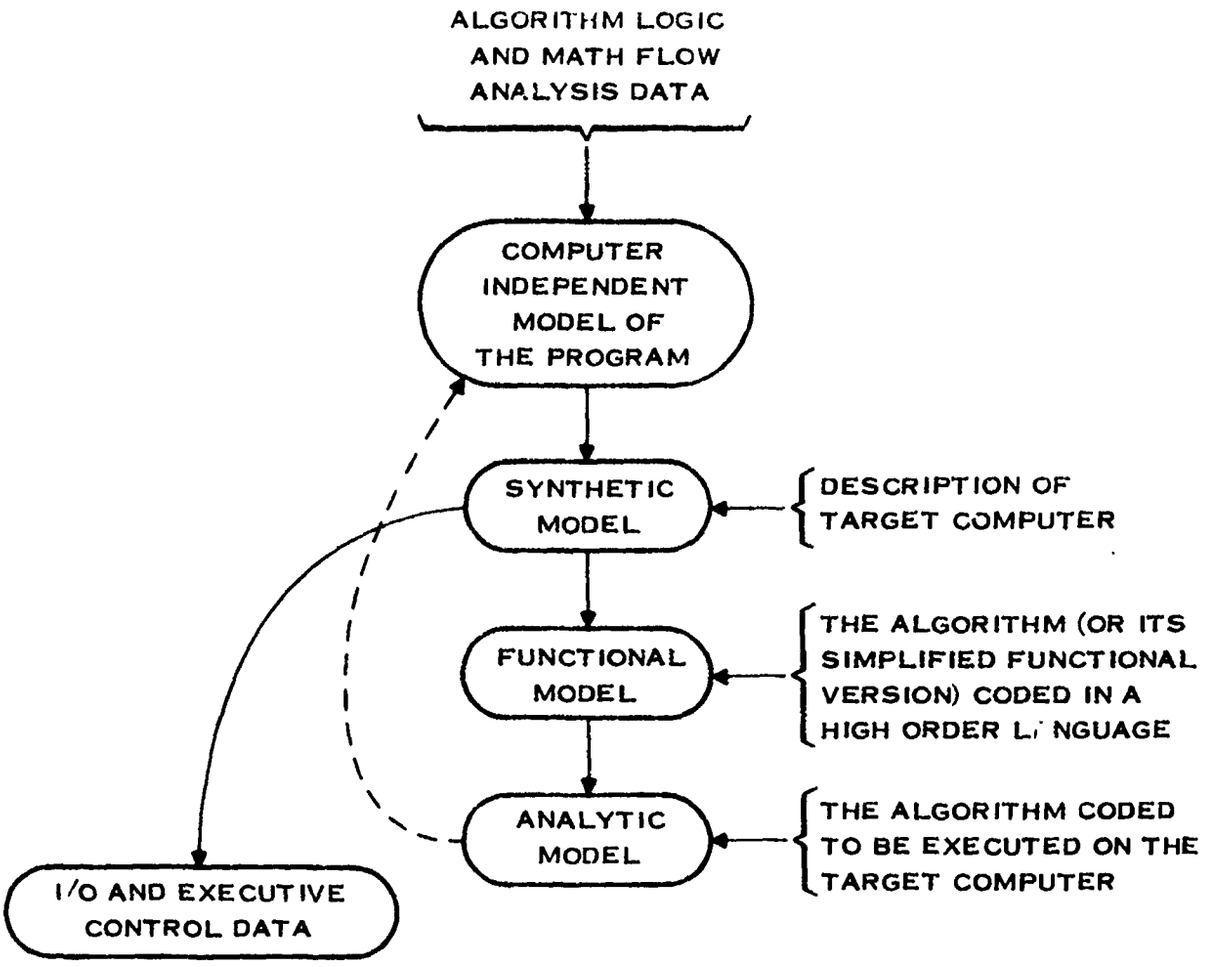



Figure 159. Evolution of Process Model

System network simulation

Functional simulation

Instruction (or Processing Element level) simulation

Hybrid analytic simulation.

The first three simulators are of discrete-stochastic type, such a simulator can be built around a common software package which is independent from the particular simulation mode, model, or the problem to which it is to be applied. This package provides a nucleus of basic subroutines for simulation.

Next follows the description of the four canonical simulation modes and of the corresponding simulators. Figure 160 shows how these four simulators are interrelated: in

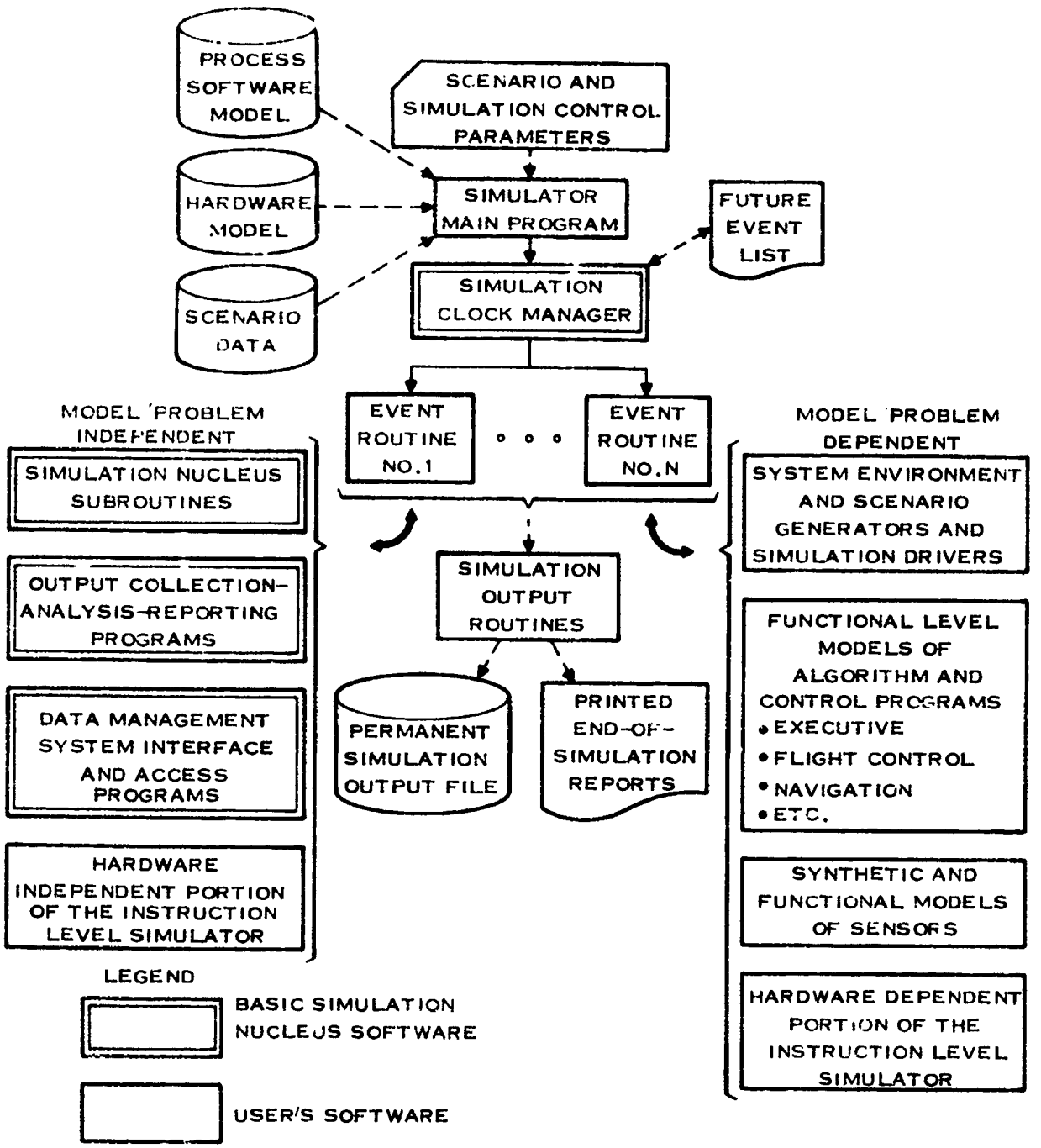



Figure 160. Overall Structure of the Relationship Between System Network, Functional and Instructional Level Simulators

particular, it shows the commonality of simulator components. Figure 161 shows the use of these four simulators in the evolutionary, top-down process development through successive simulations.

(1) System Network Simulation and Simulators

Process development starts with system network simulations on the host machine. The System Network Simulator is used as the simulation tool. The objectives of system network simulation are to

Balance the hardware and validate its sufficiency

Validate the functional design of the Executive(s)

Validate the correctness of the process control data (schedules, description of the process structure, etc.)

Evaluate the "goodness" of process structure and hardware resource allocation to process components.

The main models to be used in system network simulation are:

Process Executive(s) to be modeled functionally

The intermodule I/O and the communication with system environment (sensors and actuators) to be modeled synthetically in terms of dummy messages

Mission-oriented avionics programs to be modeled synthetically.

The simulation output data contains information on:

Hardware resource loading and usage

Information flow through the bus system

The demand on system resources by individual process components

Dynamic behavior of program execution, such as scheduling patterns, meeting deadlines, synchronization and conflict resolution between concurrently executed process components.

Avionics system engineers, computer system designers, and Executive designers are the persons principally interested in system network system simulation.

(2) Functional Simulation and Simulators

After system network simulations, the next stage in the evolutionary process development cycle is functional simulations. As noted previously, the main difference between system network and functional simulations is that the simulation of the first type models the I/O traffic through the system in terms of dummy messages (described in terms of their frequency, size, origin, and destinations), whereas the simulation of the second type in addition handles the actual value data of these I/O messages. Hence, the functional simulator must be provided with a simulation driver which accesses or generates the process (system) environment data and then pumps it into the simulation process. similarly, it needs a model of the process (system) response mechanism by means of which the process (system) reacts to and affects its own environment. For this reason, functional simulations are useful through that phase of process development during which the mission functions and algorithms (including their implementations) are tested and evaluated.

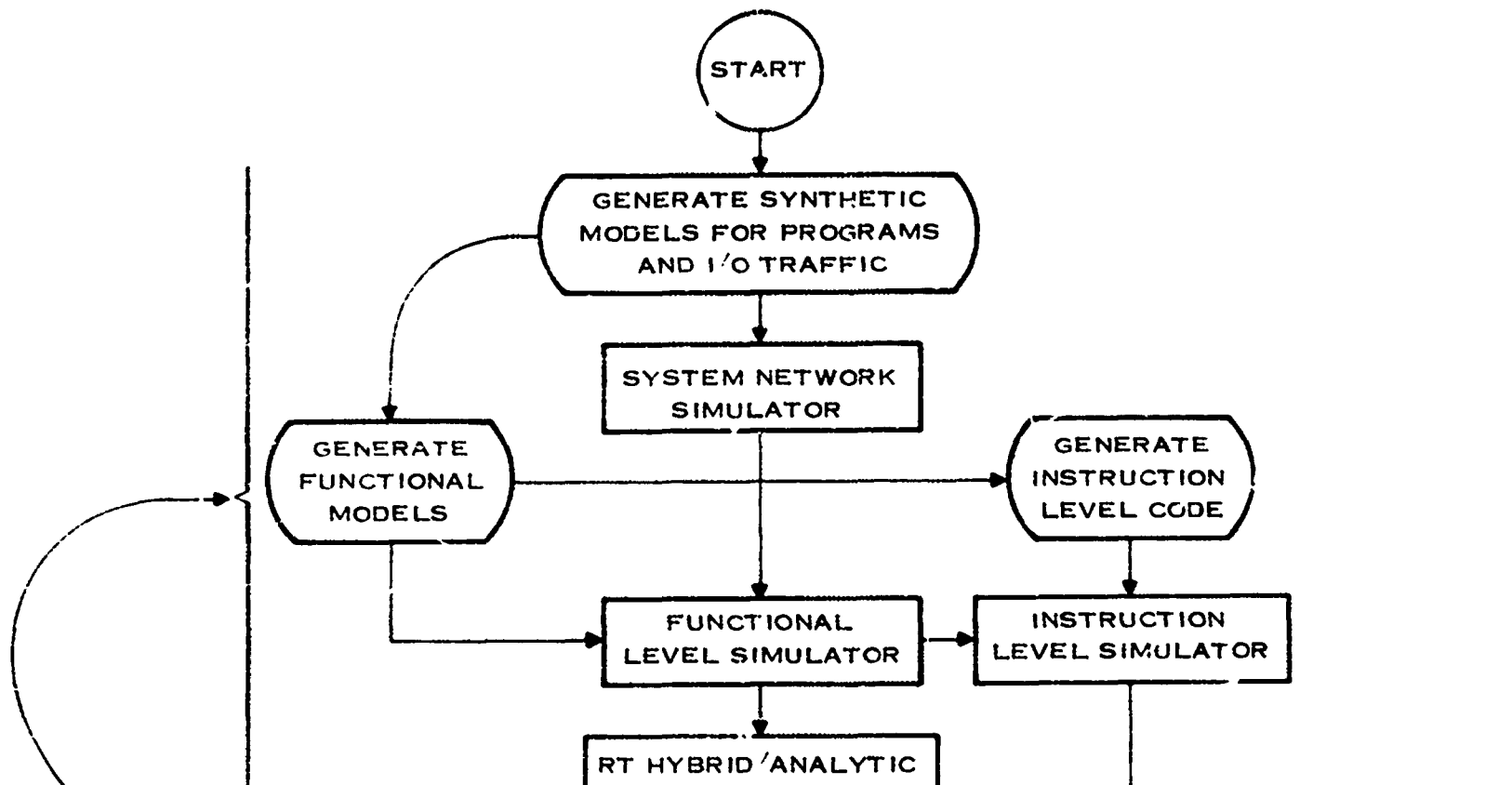



Figure 161. Use of Simulation in the Top-Down System Design Process

In addition, a processor observer can be introduced into the functional simulation structure. The observer is a mechanism whose function is to observe and record the true state of process environment (which the observer can access) and the corresponding actual inputs and outputs of the process. These recordings are done for post-simulation analysis. Thus, the mission effectiveness of the process can be measured. Such an approach allows maintenance of the desired level of the mission effectiveness of the process almost throughout its entire design and implementation cycle. Figure 162 summarizes the relationship between the process under development, its simulated environment, and the process response to the environment. It also shows the process observer, tapping information for performance evaluation from several system intercommunication links.

The main objectives of functional simulation are

- Further development of the Executive
- Development of algorithms and software for avionics functions
- Validation of program correctness and software reliability

Functional simulations are run on the host computer. The following models are used:

- Process Executive(s) to be modeled functionally, possibly with greater detail than in system network simulation
- The intermodule I/O and the communication with system environment to be modeled in part functionally (messages whose values are unknown shall be modeled synthetically)

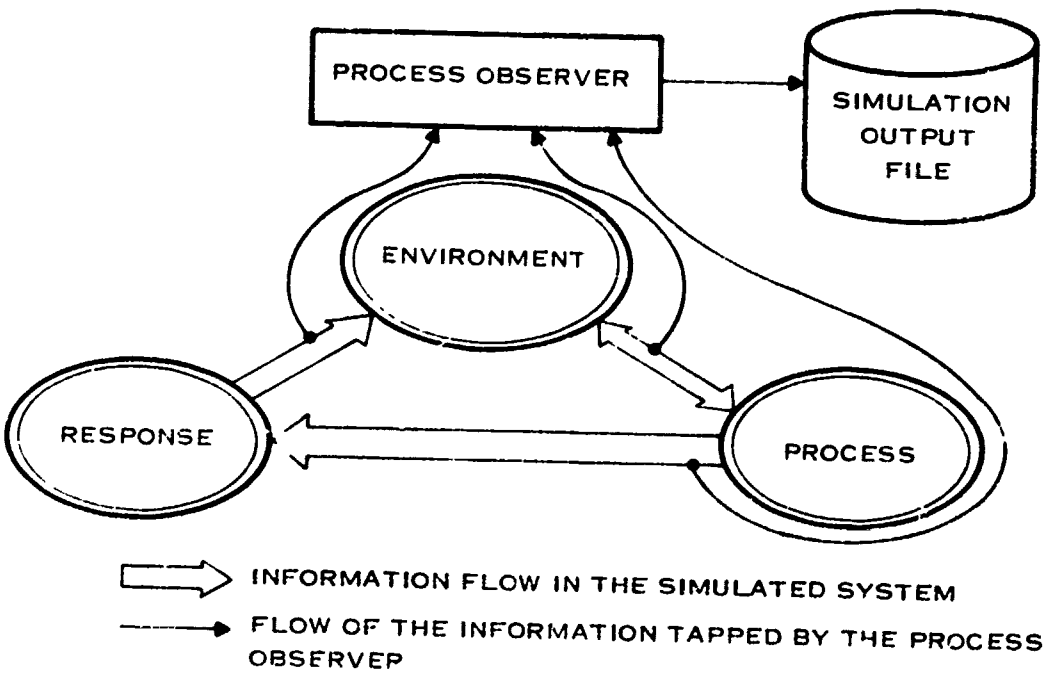



**Figure 162. Relationship Between the Process, Its Environment, and Its Response to the Environment**

Mission function programs to be modeled functionally

Process environment and response data to be modeled functionally by generating the actual values for exogenous inputs and endogenous outputs.

Functional simulation produces the following types of simulation output:

Copies of exogenous inputs into the process from its environment (such as radar returns)

Process requests for environmental data (such as radar orders)

Process response outputs (such as the process-to-actuator signals)

The process-internal functional data (such as the information being exchanged between individual functions, tasks, or other process modules)

All output data obtainable from system network simulation such as system loading, I/O traffic through it, or functional behavior of the Executive).

Note that the first three output types are observed and recorded by the Process Observer. The recorded output can be analyzed after simulation to determine the mission effectiveness performance of the process, which constitutes a capability that greatly facilitates development of mission algorithms.

(3) Instruction Level Simulation and Simulators

Simulation (simulators) of this type in DPM work is also called Processing Element simulation (simulator), for it is used to simulate the program execution on a single Processing Element (PE). A host computer is used to run instruction-level simulations. In the software process development methodology described here, the use of instruction-level simulation is limited. It is to be applied primarily to validate models for the other types of simulation and occasionally, to develop software modules for the target computer before the latter becomes available.

The process program that is to be simulated on the Instruction Level Simulator must be modeled analytically (i.e., the actual program is used). As noted previously, this program may be written in the assembly language of the target computer or preferably in a higher-order language, depending on the availability of such a language and its compiler for the target computer. Furthermore, the program does not have to represent the ultimate version of an algorithm (procedure), on the contrary, an analytic model often represents a preliminary version of the algorithm (procedure). One important characteristic of analytic models, already noted, is that such a model describes its own run-time characteristics either by being simulated on an Instruction Level Simulator or by being executed on the target machine.

The objectives of instruction level simulation in process development are:

Program validation on the host computer

Improvement of synthetic models (such as program running times)

Detailed investigations of computer architecture to fit hardware configuration to process requirements.

Instruction level simulation can produce the following outputs of interest to process design:

High fidelity data on the timing of simulated program

Detailed maps of memory usage in the simulated computer

Memory and register dumps (snapshots) of the simulated program execution

The actual outputs of the simulated program.

(4) Analytic Hybrid Simulation and Simulators

Simulation of this type is used in the late stages of process development. It runs on the target computer, i.e., on the whole DP/M network or on a representative portion of this system such as an Affinity Group. Similarly, an entire process or a representative portion of it, such as an avionics function, is simulated. Furthermore, simulated or real sensors/actuators (i.e., the process environment/response) are applied to stimulate and drive simulation which can proceed in real time or in interrupted real time (as in hot-bench test-bed simulation), depending on the configuration of the simulator hardware.

The term "analytic" in the description of this simulation denotes the fact that the analytic model of the entire process is to be used; the other term, hybrid, emphasizes that both analog as well as digital equipment may be used in the simulator hardware. The digital portion of hardware may include other computers besides the target DP/M network.

In summary, analytic/hybrid simulation will be run on the target computer (DP/M system or a substantial portion of it). Additional analog and digital equipment may be needed to drive and control the simulation process, to provide exogenous inputs for this process, and to consume the produced outputs in the closed loop with the process environment. Two types of hardware configurations, depending on whether the simulation purpose is a laboratory-like test-bed exercise or a test of the (pre)deployed system are typical:

A hot-bench test-bed or a brassboard hybrid configuration, including a driving (process environment simulation) subsystem

(Pre)deployment configuration of the target computer, including the standard test equipment.

The uses of analytic/hybrid simulation are:

Final process validation

(Pre)deployment acceptance tests

Improvement and validation of models for system network and functional simulation.

A subsystem for collecting and recording simulation output (the process observer) is still needed in this type of simulation. However, its use will have to be carefully restricted in order not to noticeably affect the real-time performance of the process under test.

## 3. Scope and Automation of Process Construction

To determine the process constructor design requirements, the following two basic issues must be decided:

The scope of functional capabilities which the process constructor is to provide

The level of automation to be built into the process construction procedure.

Any particular solution of these two problems affects the future usefulness of the process constructor. The purpose of this subsection is to summarize the approach recommended by Texas Instruments for developing the initial DD/M process construction capabilities.

*a. Scope*

As noted in Subsection IX.B.1, the scope of a process constructor is the extent of process construction functions and services that it provides. To review and illustrate the meaning of the above concept, consider two extreme cases. In the first case, the process constructor is a standard linking loader which can perform the four basic program linking and loading functions: memory space allocation, resolving symbolic references between object decks (linking), adjust all address-dependent parts of the program (relocation), and physically load instructions and data into memory. In the second case, consider an elaborate process construction system which, besides performing all functions of a simple linking loader, is capable of optimizing the process structure, determining the best hardware configuration (i.e., switching hardware to software), and generating all Executive data needed to control the process execution in real-time. There are many possible design possibilities between these two extremes. One such choice, which is similar to DP/M process construction, is shown in Figure 163.

Determination of an appropriate scope for a process constructor under development is an important design decision; it affects the cost and time required to develop the process constructor as well as its operational usefulness after it has been completed. To determine a suitable capability scope for the DP/M process constructor, the following factors were considered:

Essential functions of process construction, especially those that are peculiar to the DP/M hardware architecture and to the design of its Executive.

Features which make the use of a process constructor more convenient.

The following process construction functions have been identified to be essential:

Process Analysis: (a) decomposition of a logical (computer-independent) model of the process into computer-dependent programs such that no program exceeds the capacity of a DP/M Processing Element; (b) validation of the structural correctness of individual programs; (c) construction of the I/O traffic model

Process Synthesis: (a) derivation of an optimal hardware configuration; (b) optimal assignment of Processing Elements to programs (which can also be viewed as a mapping of individual programs into the Processing Elements of the derived hardware configuration).

Remarks:

As pointed out in Subsection IX.B.5, both optimization problems are equivalent

Optimal hardware configuration does not imply an extreme usage (saturation) of hardware resources; on the contrary, since hardware overloading reduces (re)programmability of the process software, the Process Constructor must prevent this from happening

Automatic generation of all process data needed either for process simulation or for actually running the process on the target DP/M network in real-time.

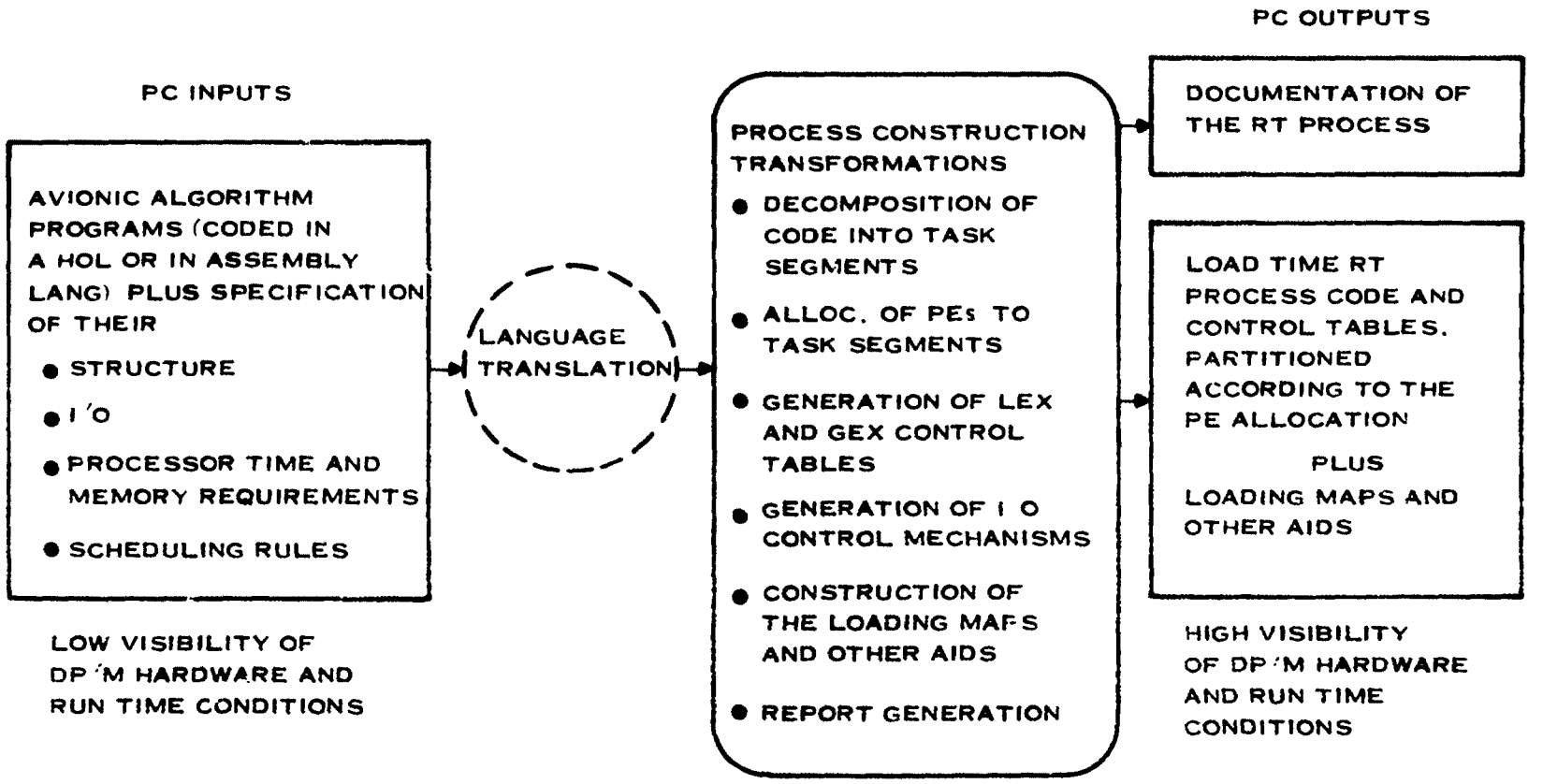



Figure 163. Process Construction of Moderate Scope

Performing all model (process) linking and loading functions for simulation on a host computer or on the target DP/M network.

Automated generation of

Process documentation

Computerized data base

Capability to modify an already constructed process by using the computerized data base preserved in process libraries

*b. Automation*

Definition of process construction automation levels determines

The function of the process designer (human) in the process construction procedure

The amount of manual work involved in such a procedure.

The purpose of this subsection is to analyze the factors that affect the automation of process construction and to suggest a recommended approach. In Honeywell's study of DP/M-like computer systems [Reference (3)], the problem of automating process construction was mainly viewed as the problem of deciding which process construction algorithms and procedures should be computerized and which one should be carried out manually. There the attention was focused on algorithms because at that time they had not yet been studied in sufficient detail to fully understand the technical and economical feasibility of their computerization. On the other hand, the automation of process construction was not viewed as primarily being a system problem, i.e., the problem of (a) providing the user with computerized information processing facilities which would minimize manual handling input and output data, (b) integrating the computerized process construction algorithms into a system of programs which would intercommunicate through a computer-resident data base, and (c) establishing an efficient man-computer interface by means of the interactive remote terminals with text editing capabilities.

In the course of the work being reported here, an overall process construction procedure for DP/M computer systems has been formulated (it is discussed in Subsection IX.C) and all key algorithms of that procedure have been identified and studied. Their analysis has led to the conclusion that:

All identified algorithms can be computerized at a reasonable cost

Some of these algorithms not only would overtax the process designer if they had to be performed manually (the main reason for this being their highly combinatoric nature) but also would be extremely prone to clerical errors, and thus would tend to make the resulting process unreliable.

To illustrate the second conclusion, consider the work involved in generating program graphs (from the input-output relations between individual process modules) and enumerating all potentially possible execution paths through that graph. Doing this manually for a small number of graphs of modest size is not an overtaxing task. However, manually analyzing a great number of such graphs in order to enumerate the execution paths and then recording the results is a

[3] Johnson, M.D., et al.: "All Semiconductor Distributed Aerospace Processor Study," Vol. II, *Technical Report AFAL-TR-73-226* (August 1973) prepared for the Air Force Avionics Laboratory, Air Force Systems Command, by Honeywell Systems and Research Division, St. Paul, Minnesota.

self-defeating effort, even the analysis of a single graph of a medium size (with more than ten execution paths) is a time consuming task. Then, there are algorithms whose manual execution is practically impossible due to the amount of combinatorics and arithmetic involved. Such algorithms occur in connection with determination of optional process structures and hardware configurations.

The considerations discussed above have led to the formulation of the following recommendations with regard to automation of DP M process construction. These recommendations are stated below by dividing them into two groups: those that are primarily concerned with the automation of process construction algorithms and those that apply to the automation of a process constructor viewed as a system in which man interfaces with computer.

The recommendations for automating algorithms are:

(1) All algorithms and procedures should be computerized, the designer's role should be limited to obtaining the input data and to examining and possibly influencing the outputs of computerized algorithms as specified below in items (2) and (3).

(2) The process designer should be reserved an option to influence, through his input data, or override the results obtained with computerized algorithms. Example: the designer should be able to specify as input to the hardware resource assignment algorithm the minimum number of Processing Elements to be present in the hardware configuration and the PE residences for a subset of process programs.

(3) The designer should be left in all decision-making nodes with more than one outgoing branch of the network representing the overall process construction procedure; i.e., he must be able to reject or accept partial or complete outputs of process construction and to determine its further course by deciding whether to proceed or to reiterate a part of the procedure.

With regard to the automation of process construction as the man-machine system, recommendations are stated in the order of priorities for various system features (which also approximately corresponds to the order of their increasing cost):

(1) Integrate the programs constituting the overall process construction procedure into a system by providing facilities for sharing the model data between individual programs (this recommendation leads to the requirement to develop a supporting computer-based information system as a part of the process constructor, the design requirements for which are stated in Section II.C).

(2) Provide programs subroutines which would generate printed reports on intermediate outputs of process construction at all designer's decision points.

(3) Provide programs subroutines which would document the process in the form of

   (a) printed reports or

   (b) the data to be stored and preserved in the libraries of DP M Process Development Information System, which has been referred to in item (1) above; the preserved model data should be organized and formatted to be directly usable as input for simulation or for execution of the process on the target DP M System.

(4) Extend the language translator which will be used to compile applications (i.e., mission-oriented) programs, or just the models of such programs, to extract and generate input descriptions of individual process modules for the process construction procedure (the purpose here is to minimize the amount of manual work needed to generate the basic inputs for process construction).

(5) Extend the DP/M Process Development Information System to include the capability to retrieve and reconfigure the information on model data stored in the libraries.

(6) Provide interactive, on-line remote terminals with text editing capabilities (including all support software/hardware) for interfacing the designer with the process constructor and other portions of the DP/M process development system.

Note that all these six features, to be added incrementally, imply the availability of the DP/M Process Development Information System. Such a system could be built gradually in parallel with the rest of functional process construction capabilities. Features (1) through (3) are sufficiently modest in cost to be implementable in the initial version of process constructor. Addition of feature (6) presents the transfer from the initial noninteractive-terminal access mode to an interactive interface between the designer and the computerized portion of process constructor, and probably represents the greatest single cost increment. In item (4) above, it is assumed that the translator of an existing language is moderately extended. If a more radical approach, involving the development of a special language and its translator is considered, it may become the most expensive and time-requiring single item.

## 4. Recommended Approach for the Development of Process Construction Capabilities

Technically, several different levels of process construction capabilities are feasible. Each is characterized by the extent of its process construction functions, the level of automation, and the amount of supporting software and hardware that it may require. Furthermore, each level is characterized by the degree of the process construction completeness that it can offer and so by its practical usefulness. At one extreme, for example, one may consider limited process construction facilities which would enable the proven designer to construct DP/M process models for simulation and whose use would require a great amount of manual effort. At the other extreme, there is a full-scale process constructor for the development and maintenance of the real-time processes which are to be put into the operational aircraft. To be at all useful, the process constructor for the last case may require extensive hardware and software support; several years may be needed to develop it and its full-scale development may involve a high degree of risk. The purpose of this subsection is to formulate an incremental approach for building up DP/M process construction capabilities which would promise limited but practically useful process construction capabilities at low cost and at an early time. This approach is low risk because the key algorithms and procedures needed in the initial process constructor are well understood presently, for they have been investigated and tested in the course of the work being reported herein.

Analysis of the technical, economic, and time factors involved has led to the formulation of an incremental, multi-step approach for building up DP/M process construction capabilities. According to this approach, full-scale process construction capabilities should be built up gradually in at least three incremental steps, the end product of each step should constitute a practically useful process constructor which would form a nucleus for the process constructor of the next step. Next, we characterize the process constructors to be developed in each of the three presently identified major steps

STEP 1 *Basic Process Constructor* whose capabilities are to be limited to construction of DP/M process models for simulation on a host developmental computer

STEP 2 *Intermediate Process Constructor*, which would be used in a controlled laboratory environment to construct (a) process models for simulation and (b) the actual software process for a target DP/M computer network.

STEP 3 *Production (or Deployment) Process Constructor* which would be applied in a typical software production and maintenance environment to facilitate (a) development of the DP/M real-time software processes which are to be used by the operational aircraft, and (b) maintenance and modifications of the processes that had already been operationally deployed.

As stated earlier, the Intermediate Process Constructor is to have all the capabilities of the initial Basic Process Constructor; the Production Process Constructor is to have all the capabilities of the Intermediate Process Constructor. Next, each above identified process constructor is discussed separately in more detail.

The reasons for starting to build up DP/M process construction capabilities with the development of Basic Process Constructor are to:

Test and evaluate the technical approach to DP/M process construction, such as the practical value and correctness of key algorithms.

Test and evaluate the system design, such as the "goodness" of the overall process construction procedure, its interface with the user, and its inputs and outputs.

Obtain initial process construction capabilities for process development based on successive simulations at the earliest time possible.

### *a. Basic Process Constructor*

As noted previously, the main algorithms and procedures of the basic process have been investigated, practically implemented, and evaluated in the course of the work which is being reported here. The conclusions that have been drawn from that work indicate that the technical problems related to the development of the Basic Processor Constructor are well understood and that such a development would be a low-cost and low-risk effort

Subsection IX.C contains a summary of requirements for the development of the Basic Processor Constructor. These requirements basically can be divided into those which specify process construction algorithms and those which are concerned with system design considerations. The analysis of process construction algorithms is discussed in the next subsection (Subsection IX.B.5). As implied in the discussion of requirements and algorithms, the Basic Process Constructor is to be highly automated from the algorithmic viewpoint, yet is automated very little from the system and the interaction-with-the-user viewpoints, i.e., it is to be only an independent set of computer programs which will intercommunicate through shared data files and which use will interface through punched input cards and provided output reports.

### *b. Intermediate Process Constructor*

The Intermediate Process Constructor is discussed next. The main purpose of developing this system is to

Test and evaluate in a laboratory environment the algorithms and system design aspects to be used in the third version of process constructor

Provide practical tools for constructing not only the simulation models but also the actual software processes, the latter are to be constructed for high fidelity exercises, such as the test bed simulations in which the actual DP M hardware would be used

As noted previously, the Intermediate Process Constructor shall possess all functional capabilities of the Basic Process Constructor and, in addition, shall be capable of constructing actual software processes for the target computer. Furthermore, the Intermediate Process Constructor should not differ much from its successor, the Production Process Constructor, in the process construction algorithms, it may differ from the Production Process Constructor in its system facilities, reliability, and efficiency. Since the Intermediate Process Constructor is primarily to be used as a test bed for algorithms and system design, it would be economically wise to provide this process constructor with an elaborate and possibly expensive, user-system interface; such as, a network of interactive terminals, each with input, output, and exit capabilities. Some of these automation features are to be introduced in the Intermediate Process Constructor in order to evaluate them for their future use in the Production Process Constructor

### *c. Production Process Constructor*

As stated earlier, the Production Process Constructor represents the end product of the process construction capability development cycle. Its main use will be construction and updating of processes for the actual field use. Hence, the main design requirements are:

The ease and simplicity of use

High real-time efficiency, reprogrammability, and robustness of the end product (operational real-time process)

Efficiency of the process construction procedure.

As noted earlier, the Production Process Constructor algorithmically should not differ much from the Intermediate Process Constructor, but it may differ from its predecessor in the level of automation of the process construction procedure.

To satisfy the first requirement stated above (the ease and simplicity of use), the Production Process Constructor shall be accessible through remote-interactive terminals. These terminals shall be capable of supporting the following system access and usage functions.

Inputting of process model and control data

Editing of data already stored in the system

Controlling the process construction procedure, overruling the process design decisions made by computerized algorithms

Displaying of intermediate and final outputs

Retrieval of the information on the process stored in system libraries.

The requirement for an efficient, reliable, and robust end product (deployable process software) is very critical because it affects not only the shape of real-time software but also the resulting hardware configuration. The requirement for real-time efficiency in process execution implies the absence of bottlenecks, delays, or missed deadlines that may affect the mission performance. The process is robust to modifications if small changes in the process software or

control cause only localized reconstruction problems: i.e., such small changes do not ripple through the entire process construction procedure and especially do not affect the originally made resource allocation assignments (and at the same time, the hardware configuration). Level of programmability can be defined as the cost of extending the software per one unit of addition. Programmability is an important economic factor in development of large-scale real-time systems due to the high cost of software relative to that of hardware [reference (4)]; it is also important because it often determines the critical time in-system development. Reprogrammability of an already operational software process is important for similar reasons.

Both the process robustness relative to modifications and its (re)programmability are associated with the level of hardware resource loading or saturation. It is well known that heavy loading levels make software difficult to (re)program. For example, if the memory is nearly exhausted, even the smallest perturbation in the code may result in a big reprogramming effort. Furthermore, relative saturation of hardware resources has a negative effect on the robustness of process. For example, doubling the scheduling rate of a program may cause exceeding the processing capacity of the PE to which the program had been originally assigned during process construction; implications of exceeding PE processing capacity are that the software process and possibly the hardware will have to be reconfigured in order to accommodate such an increase in the running rate of a program. In view of the above considerations, the process constructor must be capable of assigning hardware resources in the way that will maximize hardware load balancing (equalization) and avoid local and global hardware saturations. Thus, the second design requirement can be reworded as follows: The process constructor shall preserve a high real-time efficiency of the process (which also depends on the design of the Executive and of the individual programs) and properly match hardware to software.

By the efficiency of process construction procedure, we mean relative amounts of computer and human time required to do the job. Computer efficiency should be a lower priority requirement than are the other two requirements. In fact, it may conflict with the other two. What can be done in such a situation is fixing the acceptable performance levels with regard to the first two requirements and making the computerized process construction procedure as efficient as possible. On the other hand, human efficiency should be retained as a high priority requirement, for one of the fundamantal goals of process construction is to eliminate or minimize the manual work involved.

## 5. Analysis of Process Construction Algorithms

This section has a twofold purpose: (1) to discuss the Basic Process Constructor algorithms and procedures for which the requirements are specified in Subsection IX.C, and (2) to define a precise mathematical model for the problem of optimal allocation of hardware resources and then to discuss various approaches for solving this problem.

Algorithms of three basic types come up in process construction:

Process Analysis Algorithms these are the algorithms that are used to analyze certain aspects of process design or structure for evaluation purposes.

[4] Ramamoorthy, C.V., R.C. Cheung, and K.H. Kim: "Reliability and Integrity of Large Computer Programs," *Memorandum No. ERL-M430* (March 1974), Electronics Research Laboratory, College of Engineering, University of California, Berkley, California.

Example: the algorithm used to analyze the structural correctness of program graph (Step 4 of the process construction procedure), where a graph is said to be structurally correct if it contains precisely one entry node, precisely one exit node, and no loops through its nodes.

Process Design Algorithms they affect the overall structure and design of the process. Example: the algorithm used to allocate DP/M Processing Elements to process programs (Step 6 of the Process Construction Procedure)

Data Transformation Algorithms these are used to reduce or restructure model data and to perform format or value transformations on the data. Example: generation of control data for process simulators (Steps 7, 8, and 9 of the process construction procedure).

Note that the above given characterization of algorithms is not a classification in the sense that it is neither mutually exclusive nor all inclusive. There are algorithms in the process construction procedure which have the characteristics of all three types. For example, the first step of process construction procedure is represented by an algorithm, called Computer-Independent Model Generator, which analyzes the data to identify all input-output relations between the tasks of a program, constructs the maximally parallel predecessor-successor graph for each modeled program in the process (which is a design task), and performs various transformations on the input data in order to reorganize and compact it for future use.

### *a. Process Analysis Algorithms*

Several process analysis algorithms come up in the early phase of process construction (Subsection IX.C). These algorithms are mainly used to analyze the structure of the process and to determine the input-output precedence relations in individual programs. All process analysis algorithms can be cost-effectively computerized. Since such algorithms are not used for design decision making, there is no need to put the process designer into the loop to examine and possibly overrule their outputs. Hence, the interaction between the computerized process analysis algorithms and the designer normally can be limited to the examination of the final output data.

The following process analysis algorithms were implemented and computationally tested in the course of the Process Construction Case Study [reference (5)].

Topological Sort

Path Matrix Constructor

Execution Path Generator

They constitute the key algorithms of Step 6 of process construction procedure (Subsection IX.C). Since the above three algorithms are to be used in the Basic Process Constructor, they are described next in more detail. The following should be noted before starting their discussion:

(1) In the Experimental Process Construction System of the case study they have been embedded in a single computer program (which corresponds to the Process Analysis Program of Step 6 in the procedure discussed in Subsection IX.C) and are executed in the same order as listed above

[5] Consolver, G.A., et al.: *Distributed Processor Memory Architectures*, Interim-Technical Report for USAF Avionics Laboratory, Texas Instruments Incorporated, Dallas, Texas, September 1974.

(2) For analysis, the process is decomposed into programs and a single program is processed at a time.

(3) Assume for the following discussion that the process program to be analyzed consists of N tasks, which will be referred to as N nodes. The following items describing a process program comprise the basic input to the three algorithms:

  (a) The overall description of the program which consists of program ID, expected iteration (repetition) rate in real time, node (task) count, the IDs of the external and interprogram inputs and outputs to be handled by the program, and the ID of the entry node.

  (b) Description of each node (task), consisting of the node ID, minimum and maximum run times, permanent and temporary storage requirements, the IDs of all immediate successor nodes and probabilities of transitioning to them, a list of standard subroutines called, and the IDs of all intraprogram-intertask inputs and outputs associated with the task.

(4) One of the first steps in program analysis is to construct from the input information two fundamental matrices which are to be used in all three algorithms:

  (a) N × N transition probability matrix T, the (I,J)th element of which specifies the probability, $P_{IJ}$, for a direct transition at execution time from node I to node J. If direct transition from node I to node J is impossible, then $P_{I,J} = 0$; note that $P_{I,1} + P_{I,2} \ldots P_{I,N} = 1$ is true for each row of matrix T except the one which represents the program exit node--for that row, the probability sum is 0

  (b) N × N node adjacent matrix A, the (I,J)th element of which

$$A(I,J) = 1 \text{ if there is a direct link from node I to node J, which exists if and only if } T(I,J) = P_{I,J} > 0$$
$$= 0 \text{ otherwise.}$$

A description of the three algorithms follows.

(1) Algorithm 1: Topological Sort

This algorithm analyzes the input-output precedence relations between the nodes (tasks) of a program model and generates topologically ordered node ID numbers which are internally needed throughout the program analysis phase of process construction.

A program is topologically sorted (or its nodes are ordered) if, for any two nodes with ID numbers I and J, $I < J$ implies that the Jth node is not an immediate or remote predecessor of the Ith node.

The topological sort algorithm is discussed in references [6] and [7]. The version implemented in the process construction case study follows reference [6], in which the algorithm is referred to as Fulkerson's rule. It uses the following terminology:

[6] Berztiss, A.T.: *Data Structures: Theory and Practice*. Academic Press, Inc., New York, 1971.

[7] Knuth, D.E.: *The Art of Computing Programming*. Vol. 1 (Second Edition). Addison-Wesley Publishing Co., Reading, Mass., 1973.

(1) *Digraph* a directed graph

(2) *Acyclic Digraph* a directed graph with no closed loops (cycles) through its nodes

(3) *Indegree of a node (or of a set of nodes)* X is the number of arcs coming into X from the nodes of the digraph G that are in the complement of X with respect to G

(4) *Outdegree of a node (or of a set of nodes)* X is the number of arcs emanating from X and pointing to the nodes in the complement of X

(2) Fulkerson's Rule

Let X be the set of nodes that have been numbered (i.e., given topologically ordered IDs) at any particular stage in the application of the algorithm to an acyclic digraph. Proceed as follows.

1. Set I = 1
2. If there is no unnumbered node with zero indegree, go to 5, otherwise, go to 3.
3. Assign the number I to an unnumbered node with zero indegree as its new (topologically ordered) ID.
4. Set I = I + 1; go to 2.
5. If all nodes have been numbered (i.e., assigned topologically ordered IDs), restore all removed arcs and stop; otherwise, go to 6.
6. If the outdegree of X is not 0, remove all arcs originating from X and go to 2.

(3) Algorithm 2: Path Matrix Constructor

This algorithm analyzes the adjacency matrix of the program graph to construct the path matrix P, whose (I,J)th element,

P(I,J) = 1 if there exists at least one path from node I to node J

= 0 otherwise.

Note that the existence of a path does not imply the existence of an arc directly linking I to J, a path from I to J may traverse several nodes.

The implementation of the Path Matrix Constructor in the process construction study follows a very efficient procedure credited to S. A. Warchall, which is discussed in reference [6]. For convenience, it has been included in the sequel.

(4) Warchall's Algorithm:

Given an N X N X matrix with elements

$$X_{IJ} = 0 \text{ or } 1.$$

Proceed as follows

1 Set $X^* = X$

2 Set $J = 1$

3 Set $I = 1$

4 If $X^*_{IJ} = 1$, then set $X^*_{IK} = X_{IK} \vee X_{JK}$ for all K from 1 to N

5 Set $I = I + 1$. If $I \leq N$, then go to 4; otherwise continue

6 Set $J = J + 1$. If $J \leq N$, then go to 3; else stop

According to Warshall's theorem [ref. (6), p. 206], if X is the adjacency matrix of a directed graph G, the matrix $X^*$ generated by the above algorithm is the path matrix of G. In Step 4 above, the Boolean disjunction Operator "$\vee$" is defined as follows: $0 \vee 0 = 0$; $0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$.

Both the path matrix and the node adjacency matrix are used to analyze the following aspects of the correctness of program structure:

Absence of loops in the program graph;

Whether the designated entry node actually has no predecessors and whether there are no more nodes without predecessors;

Whether the designated exit (terminal) node actually has no successors and whether there are no more nodes without successors.

(5) Algorithm 3. Execution Path Generator

Assume that the program under analysis contains N nodes (tasks). This algorithm analyzes the N × N node adjacency matrix A of the program to construct all alternative execution paths through the program, each starting at the entry node and ending at the exit node. If there is an execution path of length K through the nodes $M_{I_1}, M_{I_2}, \ldots, M_{I_K}$ (where $M_{I_1}$ is the program entry node $M_1$, $M_{I_K}$ the exit node $M_N$), the constructed list, $M_{I_1} \rightarrow M_{I_2} \rightarrow \cdots \rightarrow M_{I_K}$, defines it. The Execution Path Generator can be considered as being a procedure for constructing the program graph from the input-output relations between individual pairs of nodes, specified by the node adjacency matrix, A. Recall that the (I,J)th element of matrix A is

A(I,J) = 1 if node I produces outputs which are consumed by node J

= 0 otherwise.

The rows (and columns) of A are assumed to be ordered by the increasing values of the node topological ID numbers.

The implementation of Execution Path Generator algorithm in the Process Construction Case Study [reference (5)] is based on the following principles:

Start at the entry node to initialize a path for each outgoing arc (branch successor), add the ID number of the successor for each initialized path.

Proceed from node J-1 to node J (or equivalently from the (J-1)th to the Jth row of the node adjacency Matrix A), where the node ID numbers correspond to the topological ordering of nodes constructed by Algorithm 1

If the Jth row of A represents the last (terminal) node in an already partially completed path, extend this path by adding to it the node ID number K which corresponds to the leftmost nonzero entry in the Jth row of A. Then continue to the right in the same row of A and, for each newly encountered nonzero entry, e.g., A(J,L), create a new path by

Copying the first (leftmost) path that was extended in row J

Deleting from it the lastly added node (whose ID number is K)

Adding to it the node ID number corresponding to the Lth column

Stop after processing the (N-1)th row of A (the last row represents the exit node)

Computer implementation of this algorithm is greatly simplified if list processing facilities are available to the programmer, for the number of paths to be newly added or lists if each list represents a path tends to increase as one progressively steps through the rows of the adjacency matrix A.

*b. Process Analysis Program*

In the Process Construction Case Study, algorithms 1, 2, and 3 constitute parts of a bigger program, called the Process Analysis Program. Since the Process Analysis Program has been found to be a useful tool for analyzing process decomposition into programs (i.e., to determine whether no program exceeds the capacity of a single DP/M Processing Element) and since a program similar to it has been specified at Step 4 in the process construction procedure of the Basic Process Constructor (Subsection IX.C), it is outlined below.

The Process Analysis Program represents a multi-step procedure consisting of the following main steps.

1. Read the input description of a program and, using Algorithm 1, generate the topologically ordered node (task) ID numbers.
2. Generate the node adjacency matrix A and the branch transition probability matrix T (in both matrices, rows and columns must be ordered by the increasing values of the topological ID numbers; i.e., if I is such a number, then the node which is represented by I must correspond to the Ith row or column of matrices A and T)
3. Using Algorithm 2, generate the path matrix, P.
4. Analyze the structural correctness of the program under analysis; print a report on results of this analysis.
5. Apply Algorithm 3 to construct all alternative execution paths through the program.
6. For each path, compute its execution probability and projected execution timing.
7. By considering the specified iteration rate of the program, compute the loading of the processor timeline in real time by the program; compute the permanent and temporary storage required by the program.
8. Print a report summarizing the results of preceding steps.

To illustrate the above procedure, the output reports produced by the present implementation of the Process Analysis Program are shown in Figures 164 through 170. Two basic reports, Program Analysis Report (Figures 164 through 168) and Program (Synthetic) Model Report, are included. Note that these reports are similar to those to be produced in Step 4 of the process construction procedure which is described in Subsection IX.C.

The function of Program Analysis Report is to provide the process designer with the information which will enable him to decide whether the program satisfies all structural requirements. This report consists of five pages. The first page (Figure 164) contains a summary of the input information which describes the program under consideration. The last page (Figure 168) shows the results of the structural analysis of the program. In the particular case shown, the program whose name is NARBSWD has been found to be structurally correct.

The second report (Figures 169 and 170) summarizes the synthetic model of the program. Its function is to provide the proven designer with the information needed to decide whether the program in question can be run on a single Processing Element. If the answer is yes for all programs in the process model, we say that it is properly decomposed.

### *c. Optimal Allocation of Hardware Resources to Process Components*

This subsection formulates a mathematical optimization model for the hardware resource allocation problem and then examines alternative approaches for solving it. We start with a background review, state a few assumptions to establish firm working rules, and then proceed with the problem statement. Next, we suggest several alternative algorithms for solving this problem.

The function of the Hardware Resource Allocation Algorithm is to allocate hardware resources (i.e., the DP/M Processing Elements) to process programs. This assignment is to be optimal in the sense that it is to minimize the resulting data bus traffic, which also amounts to minimizing the number of Processing Elements in a DP/M network. Hence, this algorithm can be considered as being a procedure for determining the best hardware configuration for a given model of the real-time process. Before we proceed with the discussion of the Hardware Resource Allocation Algorithm, we shall elaborate the idea, stated at the end of the preceding paragraph, about fitting hardware to software in designing real-time systems. In the traditional approach of real-time system development, the software development effort has to wait until the hardware is selected and then the programs are written under hardware constraints; the important consequence of this approach is that the development and the final shape of mission-oriented algorithms also become subject to hardware constraints. Although this approach has many disadvantages, it could be somewhat justified at the time when the hardware cost was the dominating factor in computer system development and when the current techniques and tools, such as simulation, for developing real-time software on a host computer were not available; but the situation has been reversed during the last decade by a sharp decrease in hardware cost relative to that of software. For example in reference [4], the cost of software amounted only about to 25 percent of the USAF budget for electronic data processing in 1960 (75 percent went for hardware), while in 1973 the cost of software amounted to approximately 80 percent of the USAF budget for EDP. According to the reference just cited, the traditional real-time system development philosophy based on fitting software to hardware suffers from the following disadvantages:

DP/M PROCESS CONSTRUCTION DATE: 750116

PROGRAM NAME: NARBSWD
ID #: 0
# OF ITERATIONS PER SECOND: 32

# OF NODES: 9
INPUTTED ID # OF ENTRY NODE: 184

COMPUTER MODEL: DP/M PROTOTYPE

| NODE ID# | NODE NAME | TOPO ID# | PERM MEM. | TEMP MEM. | MIN REP# | MAX REP# | MIN RUN T(USECS) | MAX RUN T(USECS) | IN-DEGR | SUCCESSOR ID# | T.PROB |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 184 | NBEN | 1 | 18 | 39 | 1 | 1 | 32 | 32 | 0 | 185 | 0.2500 |
| | | | | | | | | | | 189 | 0.7500 |
| 185 | NBIN | 2 | 178 | 7 | 1 | 1 | 1359 | 1365 | 1 | 186 | 0.5000 |
| | | | | | | | | | | 187 | 0.5000 |
| 186 | NBAR | 3 | 156 | 6 | 1 | 1 | 2973 | 2981 | 1 | 188 | 1.0000 |
| 187 | NBLA | 4 | 883 | 36 | 1 | 1 | 5471 | 5507 | 1 | 188 | 1.0000 |
| 188 | NBPL | 5 | 73 | 3 | 1 | 1 | 3223 | 3231 | 2 | 189 | 1.0000 |
| 189 | NBLS | 6 | 279 | 10 | 1 | 1 | 2297 | 2303 | 2 | 190 | 1.0000 |
| 190 | NBOT | 7 | 120 | 0 | 2 | 2 | 210 | 210 | 1 | 191 | 1.0000 |
| 191 | NBRP | 8 | 960 | 26 | 1 | 1 | 1979 | 2043 | 1 | 192 | 1.0000 |
| 192 | NBHO | 9 | 483 | 19 | 1 | 1 | 2548 | 2568 | 1 | | |

Figure 164. Program Analysis Report, Page 1

Inability to start the software development until hardware has been procured, or at least selected, pushes software further out in the critical path

Since hardware selection must be made without understanding the actual requirements imposed by the software process, selection of an inadequate hardware configuration may dramatically affect the productivity of software development

The resulting hardware selection is poor functionally and is cost-ineffective.

As we approach about 85 percent usage of processor time and memory, the software cost and the time required for software development abruptly curve up to infinity. Further, the cost

DP/M PROCESS CONSTRUCTION DATE: 750116

PROGRAM NAME: NARBSWD
ID #: 0
# OF ITERATIONS PER SECOND: 32

# OF NODES: 9
INPUTTED ID # OF ENTRY NODE: 184

COMPUTER MODEL: DP/M PROTOTYPE

TRANS PROB MATRIX TPROBX

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.0000 | 0.2500 | 0.0000 | 0.0000 | 0.0000 | 0.7500 | 0.0000 | 0.0000 | 0.0000 |
| 2 | 0.0000 | 0.0000 | 0.5000 | 0.5000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 3 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 4 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 5 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.0000 | 0.0000 | 0.0000 | 0.0000 |
| 6 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.0000 | 0.0000 | 0.0000 |
| 7 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.0000 | 0.0000 |
| 8 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.0000 |
| 9 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |

Figure 165. Program Analysis Report, Page 2

DP/M PROCESS CONSTRUCTION DATE: 750116

PROGRAM NAME: NARBSWD
ID #: 0
# OF ITERATIONS PER SECOND: 32

# OF NODES: 9
INPUTTED ID # OF ENTRY NODE: 184

COMPUTER MODEL: DP/M PROTOTYPE

NODE ACJACENCY MATRIX NAMX

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 166. Program Analysis Report, Page 3

DP/M PROCESS CONSTRUCTION DATE: 750116

PROGRAM NAME: NARBSWD
ID #: 0
# OF ITERATIONS PER SECOND: 32

# OF NODES: 9
INPUTTED ID # OF ENTRY NODE: 184

COMPUTER MODEL: DP/M PROTOTYPE

PATH MATRIX PMX

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 167. Program Analysis Report, Page 4

```
**********************************************************************

                    DP/M PROCESS CONSTRUCTION          DATE: 750116

          PROGRAM NAME:                                      NARBSWD
                  ID #:                                            0
          # OF ITERATIONS PER SECOND:                             32

          # OF NODES:                                             9
          INPUTTED ID # OF ENTRY NODE:                        184

          COMPUTER MODEL:                                DP/M PROTOTYPE
**********************************************************************

INPUTTED INFORMATION :

N =      9

ITS =    1

ITX =    9


RESULTS OF THE TEST ON THE CORRECTNESS OF GRAPH STRUCTURE

ALL NODE ID #S ARE TOPOID #S

NO. OF CYCLES                                        =  0     (OK IF ZERO)

ID OF THE STARTING NODE                              =  1     (OK IF =ITS)
NO. OF NODES NONREACHABLE FROM ITS                   =  1     (OK IF =1)
ID OF THE EXIT NODE                                  =  9     (OK IF =ITX)
NO. OF NODES FROM WHICH THERE IS NO PATH TO ITX      =  1     (OK IF =1)

**********************************************************************
```

**Figure 168. Program Analysis Report, Page 5**

to maintain and modify the software process which hardly fits a computer is economically prohibitive. (It is ironic that, in spite of what just was been said, some system designers continue to be proud of being capable of tight designs.) With decreasing cost of hardware and with rising cost of software, getting too close to the saturation points of the designs must by all means be avoided. This consideration must be kept in mind in order to prevent the misuse of the Optimal Hardware Resource Allocation Algorithm; although it will allow matching hardware to software, the resulting hardware configuration must not be allowed to become too tight. Actually, the algorithm must be designed to permit its user to specify and control the desired level of hardware saturation.

DP/M PROCESS CONSTRUCTION DATE: 750116

PROGRAM NAME: NARBSWD
ID #: 0
# OF ITERATIONS PER SECOND: 32

# OF NODES: 9
INPUTTED ID # OF ENTRY NODE: 184

COMPUTER MODEL: DP/M PROTOTYPE

| NODE ID# | NODE NAME | TOPO ID# | PERM MEM. | TEMP MEM. | MIN REP# | MAX REP# | MIN RUN T(USECS) | MAX RUN T(USECS) | IN-DEGR | SUCCESSOR ID# | T.PROB |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 184 | NBEN | 1 | 18 | 39 | 1 | 1 | 32 | 32 | 0 | 185 | 0.2500 |
| | | | | | | | | | | 189 | 0.7500 |
| 185 | NBIN | 2 | 178 | 7 | 1 | 1 | 1359 | 1365 | 1 | 186 | 0.5000 |
| | | | | | | | | | | 187 | 0.5000 |
| 186 | NBAR | 3 | 156 | 6 | 1 | 1 | 2973 | 2981 | 1 | 188 | 1.0000 |
| 187 | NBLA | 4 | 883 | 36 | 1 | 1 | 5471 | 5507 | 1 | 188 | 1.0000 |
| 188 | NBPL | 5 | 73 | 3 | 1 | 1 | 3223 | 3231 | 2 | 189 | 1.0000 |
| 189 | NBLS | 6 | 270 | 10 | 1 | 1 | 2297 | 2303 | 2 | 190 | 1.0000 |
| 190 | NBOT | 7 | 120 | 0 | 2 | 2 | 210 | 210 | 1 | 191 | 1.0000 |
| 191 | NBRP | 8 | 960 | 26 | 1 | 1 | 1979 | 2043 | 1 | 192 | 1.0000 |
| 192 | NBHD | 9 | 483 | 19 | 1 | 1 | 2548 | 2568 | 1 | | |

**Figure 169. Program Model Report, Page 1**

To formulate the hardware resource allocation problem and the optimality criterion for this problem, the following assumptions have been made.

No program in the process model exceeds the specified level of the total capacity of a single PE. (As the process construction progresses beyond the process decomposition analysis, we are guaranteed that this will be the case.)

Selected DP/M Processing Elements may have to be preassigned to certain programs, which still may leave enough processing capacity in these PEs to assign them also to other programs.

The hardware resource allocation algorithm to be discussed in the sequel must be capable of being applied either to the entire process or to the individual

```
                    DP/M PROCESS CONSTRUCTION                DATE: 750116


          PROGRAM NAME:                                          NARBSWD
                ID #:                                                  0
          # OF ITERATIONS PER SECOND:                                32

          # OF NODES:                                                 9
          INPUTTED ID # of ENTRY NODE:                             184

          COMPUTER MODEL:                                  DP/M PROTOTYPE

# OF EXECUTION PATHS =                                               3

PATH
  #     PROB MIN RUN T MAX RUN T    PATH DESCRIPTION
-------------------------------------------------------------------------

  1 0.12500    14831.0    14943.0  184->185->186->188->189->190->191->192
  2 0.75000     7276.0     7366.0  184->189->190->191->192
  3 0.12500    17329.0    17469.0  184->185->187->188->189->190->191->192

-------------------------------------------------------------------------

PROCESSOR LOADING  =  5.590080E-01 SECS/SEC  OR  = 55.90 % OF CAPACITY


MEMORY OCCUPANCY
          PERMANENT  = 3141 16-BIT WORDS
          TEMPORARY  =   39 16-BIT WORDS
```

**Figure 170. Program Model Report, Page 2**

components of the process, such as various avionics subfunctions (The latter option may be needed to guarantee a desired number of distinct Processing Element Affinity Groups in the hardware configuration.)

The second assumption has been made to guarantee the placement of External I/O Handlers (programs especially designed to handle the external I/O) within the Processing Elements which are connected with appropriate sensors or actuators. It can also be used to guarantee a desired minimum number of Processing Elements in the entire system or just in an Affinity Group of Processing Elements.

The hardware resource allocation procedure is mainly concerned with minimizing the interprogram bus I/O traffic; the assignments to be made cannot affect the intraprogram-intertask I/O traffic because of the requirement that a program always be mapped into a single PE. However, the assignments indirectly affect the amount of bus traffic that results from the external I/O due to communications between the process components and various sensors or actuators. Recall that all external I/O is to be channeled through special external I/O handling programs, already referred to as External I/O Handler , which must be placed in preassigned Processing Elements, i.e., in those PEs which are directly connected to appropriate sensors or actuators. To other programs of the process, an External I/O Handler looks just as another program: if the handler is communicating with a sensor, it may produce outputs that are consumed by the avionics programs as their inputs; furthermore, these communications may or may not cause bus traffic, depending on the locations of the consumer programs relative to the sensor. Similarly, if a handler interfaces with an actuator, it receives inputs, sent to it by various avionics programs as their outputs, which again may or may not generate bus traffic. Thus the relative locations of an External I/O Handler program and the avionics programs that communicate with the former determine the amount of the resulting bus traffic. The Hardware Resource Allocation Algorithm attempts to minimize the potential traffic by coalescing (within the given constraints) heavily intercommunicating programs into the same Processing Elements.

The problem of optimal hardware resource allocation for DP/M processes can be stated as follows:

(1) Let

$N_{PE}$ = The initial number of PEs in the system (or in the Affinity Group under consideration)

$N_T$ = Number of programs

$T_I$ = The processor time required per one unit of real time, such as 1 second, by the Ith program

$S_I$ = The amount of storage needed for the Ith program

$M_{IJ}$ = Amount of I/O traffic between programs I and J per one unit of real time.

(2) Consider the allocation variables $X_{IK}$, where

$X_{IK}$ = 1 if the Kth PE is allocated for program I

= 0 otherwise.

(3) It will be convenient to introduce vectors

$$X_I = (X_{I1}, X_{I2}, \ldots, X_{IK}, \ldots, X_{INPE})$$

Then the dot product

$X_I \cdot X_J$ = 1 if tasks I and J are put into the same PE

= 0 otherwise

(Note that the notation used above neglects to denote vector transposes.)

(4) We want to find the maximum of

$$Q = \sum_{I=1}^{N_T} \sum_{I>J}^{N_T} M_{IJ} \ (X_I \cdot X_J)$$

subject to the following constraints for K = 1, 2, . . . . $N_{Pt}$ :

$$\sum_{I=1}^{N_T} T_I \ X_{IK} \ < E_T \text{) seconds}$$

$$\sum_{I=1}^{N_T} S_I \ X_{IK} < \text{(Number of words available in memory of Kth PE for avionics programs)} \times E_S$$

where $E_T$ and $E_S$ are the desired upper limits on the processor time and storage saturation levels, respectively; clearly, the bounds, $0 < E_T < 1$ and $0 < E_S < 1$, must be satisfied.

The following remarks apply:

Performance index Q measures the amount of bus traffic eliminated by making, within the stated constraints, the most heavily intercommunicating programs share the same Processing Elements.

To simplify the optimization problem, the amount of the processor time gained by eliminating messages from the bus traffic has not been accounted for in the constraints on processor time; the effect of this simplification is to make the constraints slightly pessimistic.

Solving this problem also solves the problem of determining the minimum number of PEs needed for the considered portion of process. Initially, set $N_{Pt}$ (= number of PEs) large, perhaps equal to the number of programs in the model. The obtained optimal values for the assignment variables, $X_{IK}$ will show which PEs will remain unused; these PEs should be eliminated from the hardware configuration.

Since certain programs (expecially External I/O Handlers) must be mapped into predetermined PEs, their assignment variables, $X_{IK}$ , are *a priori* known. Thus, the solution of the optimization problem is reducible to finding optimal values for the remaining $X_{IK}$ , which in the literature on optimization are known as free variables, in contrast to the *a priori* set or fixed variables.

The optimization problem as stated above is a quadratic integer programming problem; since

$$X_I \cdot X_J = 1 \text{ or } 0 \text{ for any I and J}$$

this program is reducible to a standard linear integer programming problem.

Several algorithms for solving the linear integer programming problem are known in literature [references (8) and (9)]. However, these algorithms are computationally expensive and typically require a large amount of memory. Therefore, it was decided to look into faster but suboptimal heuristic procedures. A class of such procedures is represented by what is known as cluster analysis methods; these techniques have found applications in many fields (such as pattern recognition, information retrieval, etc.) where large quantities of objects are grouped by constructing clusters, each containing objects interrelated by certain criteria.

In our case, a cluster represents a group of programs that heavily intercommunicate among themselves and relatively little with the programs belonging to other clusters. Furthermore, if the time and memory constraints are satisfied by a cluster of programs, a single PE may be associated with it. At the start of a clustering procedure, all External I/O Handlers must be preassigned as seeds of clusters (depending on sensor or actuator locations, several External I/O Handlers may form a single seed). In addition, other (i.e., non-I/O) programs may be designated as cluster seeds. As programs are assigned to or transferred between already established clusters during the execution of a clustering algorithm, individual clusters may exhaust the capacity of the DP/M processing elements; several options exist for what to do in such a situation. One approach is to proceed without paying any attention to the constraint violations until all programs have been clustered; thereafter, split each violating cluster into several clusters. Another strategy is to create new clusters just before a constraint violation occurs.

To apply the cluster analysis techniques to the DP/M hardware resource allocation problem, the mathematical model of the optimization problem outlined above must be expanded to include the measure of association between programs. Many different measures are known in technical literature [(10) is a good first reference]; furthermore, each measure can be used with many different clustering procedures. Selection of a proper measure and of suitable measurement variables is important, for it may critically affect the separability of programs into clusters.

To see how association between programs can be measured for the DP/M hardware resource allocation problem, consider the following example, which illustrates one possible approach:

Define the $N_T \times N_M$ program-to-I/O matrix P, where $N_T$ = number of programs and $N_M$ = number of the interprogram I/O messages (types of inputs/outputs) in the process model, as follows:

$$P(I,K) = \pm M_K \quad \text{if the Ith program produces (or consumes) the Kth message}$$

$$= 0 \quad \text{otherwise}$$

[8] Edwards, J. "Hierarchial Control in Operating Systems for Multiple Computer Systems," Ph.D thesis (in preparation), the University of Oklahoma, Norman, Oklahoma.

[9] Garfinkel, R. S., and G. L. Nemhauser: "Integer Programming," John Wiley and Sons, Inc., New York, 1972.

[10] Anderberg, M. R.: "Cluster Analysis for Applications," Academic Press, Inc., New York, 1973.

Here, the quantity $M_K$ (>0) measures the bus loading per unit time by the Kth message; this loading is proportional to the message length in bits and to the expected transmission rate.

The following two related measures of association between two programs are examined in reference [8] with respect to their potential usefulness in the hardware resource allocation algorithm:

$$D_1(I,J) = \frac{\sum_{K=1}^{N_M} \left| |P(I,K)| - |P(J,K)| \right|}{\sum_{K=1}^{N_M} \left| |P(I,K)| + |P(J,K)| \right|}$$

and

$$D_2(I,J) = \sum_{K=1}^{N_M} \left[ \frac{\left| |P(I,K)| - |P(J,K)| \right|}{(|P(I,K)| + |P(J,K)| + 1)} \right]$$

The following remarks apply:

Note that both measures make sense only for non-negative data. This is the reason for using only the absolute values of the elements of matrix P in $D_1$ and $D_2$.

Measure $D_2$ should normally be used for binary data only.

$D_1$ and $D_2$ measure the degree of agreement between two programs; i.e., if two programs, I and J, use the Kth message, weighted by $M = |P(I,K)| = |P(J,K)|$, for communication, then the corresponding term in the numerator of $D_1$ (or $D_2$) is zero, with an opposite effect occurring in the denominator. The more similar two programs are, the smaller will be the value of $D_1$ or $D_2$.

During a systematic investigation of the optimization problem stated above, attention was focused on the following four alternative methods for solving the optimal hardware resource allocation problem:

(1) An enumerative integer programming method, called Branch and Bound Algorithm [reference (9), Chapter 4]

(2) A cluster analysis algorithm known as MacQueen's K-Means Method [reference (10), Chapter 7]

(3) A cluster analysis algorithm, known as Forgy's Method [reference (10), Chapter 7]

(4) A heuristic method, which is presented in the sequel as Algorithm 4.

Preliminary computational results indicate the superiority of MacQueen's Method over methods 3 and 4 for the problem under consideration. Therefore, further research is likely to be narrowed down to methods 1 and 2. To make method 1 (Branch and Bound Algorithm) computationally more attractive, bit string manipulation techniques have been used to handle the enumeration tree on which the method is based. As stated earlier, reference [8] discusses the results of this evaluation effort and contains a detailed description of the investigated algorithms. (Although the problem addressed in reference [8] differs slightly from the optimal resource allocation problem for DP/M, the results generally are applicable to the DP/M case.) Since method 4 has been formulated in the course of DP/M process construction studies [actually, it constitutes an extension of a procedure mentioned in reference (3)], and may not be easily available in published literature, its definition has been included in the present report. Its main virtue is simplicity. It will be referred to as Algorithm 4D; methods 1, 2, and 3 will be referred to as Algorithms 4A, 4B, and 4C, respectively.

ALGORITHM 4D: SUBOPTIMAL RESOURCE ALLOCATION ALGORITHM

Let

$N_{PE}$ = Number of Processing Elements under consideration

$N_P$ = Number of programs under consideration.

Then proceed as follows:

1. Initially set $N_{PE} = N_P$. Assign one program per PE. Mark all pairs of PEs eligible for fusion. Construct the list of all eligible pairs.
2. Find an eligible pair of PEs whose fusion into a single PE (i.e., coalescence of the programs resident on these two PEs into a single PE) would eliminate the greatest amount of bus traffic.
3. Can the load (memory and processing speed requirements) already assigned to the PEs of the pair considered for fusion be handled by a single PE? If yes, go to 4: otherwise, go to 5.
4. Fuse the PEs of the candidate pair into a single PE and coalesce their workloads. Decrease $N_{PE}$ by 1. Modify the list of eligible pairs. Go to 2.
5. Eliminate the pair that cannot be fused from the list of eligible pairs.
6. Are there any eligible pairs left? If yes, pick one and go to 2; otherwise, stop.

### *d. Identified But Not Implemented Process Analysis Algorithms*

There is only one key process analysis algorithm which has been identified in the process construction procedure of Basic Process Constructor (Subsection IX.C) but was not implemented. Its function is to construct the maximally parallel, determinate predecessor-successor graphs for each program in the process model. A maximally parallel graph that generates the program determinacy is one which maximizes the potential parallelism in the program execution patterns without jeopardizing the determinacy of computed outputs. An execution pattern here refers to a sequence in which the program tasks (nodes) are to be executed. The input data needed to construct a maximally-parallel, determinate predecessor-successor graph for a program is the information on input-output relations between the tasks constituting the program. Once this information is available, many such graphs can be constructed for a given system of tasks, all of

which are logically equivalent but different in complexity (i.e., they differ in the number of links that directly connect pairs of nodes). It can be proved that every program possesses a unique maximally-parallel, determinate predecessor-successor graph of minimum complexity. References [11], [12], and [13] discuss parallel predecessor-successor graphs, including their construction.

## C. SUMMARY OF REQUIREMENTS

This subsection summarizes requirements for the development of an initial version of the system for constructing DP/M real-time processes. This initial system is called the Basic Processor Constructor (BPC). Various problems associated with its development have already been discussed in the preceding section (Subsection IX.B). Although an attempt has been made to keep this summary of BPC development requirements self-contained at the cost of restating a few basic concepts and definitions already introduced in the problem analysis section, certain references to that section, especially to the subsection which discusses process construction algorithms, could not be completely avoided for reasons of conciseness.

The requirements presented in the sequel have been divided into three major groups:

System requirements

Functional capability requirements

Implementational requirements.

The system requirements view the Basic Process Constructor as an integrated system of computer programs which interfaces with its main user, the process designer. The basic process construction procedure defined in Subsection IX.C.5 assumes the process designer's active participation in that procedure, which occurs through:

Provision of process construction inputs (mainly the model data) by the designer

Monitoring of intermediate results

Optional intervention in the computerized procedure in order to influence its further progress.

The computer programs constituting the BPC intercommunicate through shared data files; i.e., the intermediate outputs produced by one program are used as inputs to other programs. The final outputs of a process constructor are to be used as inputs either for process simulation on a host computer or else for executing the process on a target DP/M system.

BPC system requirements cover the following areas:

The interaction of process construction with other phases of process development and the interface of BPC with other systems (Subsection IX.C.2)

The assumptions made about the process development language (Subsection IX.C.3)

[11] Coffman, Jr., E. G., and P. J. Denning: *Operating Systems Theory*, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1973.

[12] Bernstein, A. J.: "Analysis of Programs for Parallel Processing," *IEEE Trans. Comp. EC-15*, #5 October 1966), pp 757-762.

[13] Baer, J. L.: "A Survey of Some Theoretical Aspects of Multiprocessing," *ACM Computing Surveys*, 5, #1 (March 1973).

A supporting subsystem, called the Process Development Information System (PDIS), whose function is to facilitate handling of the process development data base (Subsection IX.C.4).

The functional capability requirements define the process construction which the BPC must be capable of accomplishing. These requirements are stated by specifying a fundamental process construction procedure which consists of the following major steps/actions:

Analysis and reduction of the process construction input data

Process decomposition analysis, whose functions are to

Decompose a computer-independent model of the process into individual DP/M programs so that none of them will exceed the real-time capacity of a single DP/M processing element

Validate the structural correctness of each program

Construct computer-dependent program models.

Process synthesis, whose main function is to determine an optimal hardware configuration by obtaining the best possible mapping of programs into Processing Elements.

Generation of the process model data for simulation or for actual execution in real time.

The functional capability requirements are stated in Subsections IX.C.5 and IX.C.6; the first of these two sections defines the fundamental process construction procedure of the BPC, and the second one summarizes the algorithms and the external inputs/outputs of that procedure. For several reasons, no attempt was made to specify the detailed organization and formats of the data sets or files that have been identified in the fundamental process construction procedure. For example, the file organization and formats depend on the selection of the host computer on which the BPC is to be implemented; and also, it is not known at the present time in what form the process construction inputs will come (which to some extent depends on the capabilities of the process development language processor) and precisely what all output data should be (it depends on the final design of the DP/M real-time executive).

The few implementational requirements that could be identified at the present time define the expected minimum capabilities of the host computer system. They are summarized in Subsection IX.C.7.

## 1. Assumptions

This subsection contains a summary of the basic assumptions about the DP/M hardware and software from which the DP/M process construction requirements have been derived. For convenience, these assumptions have been separated into those on hardware, software process, and the process dynamics at exectuion time.

### *a. Hardware*

In a DP/M network, the microprocessors, called processing elements (PEs), are grouped into one or several clusters (Affinity Groups). All PEs within a single cluster communicate

through a Local bus. A Global bus is used to support communication between any two PEs that are located in two different clusters. The exact configuration of each cluster is determined at process construction time by the avionics functions that have been assigned to it and by the external I/O that it must service.

A PE can be characterized as a 16-bit parallel microcomputer with a 4K to 8K memory and the computational power of 250K operations/second. High-density, low-power LSI technology is to be used to implement the hardware.

Any PE can be made to communicate with the external world (i.e., sensors and actuators) through a direct parallel I/O channel, each of which is dedicated to a particular function. Note that a sensor or an actuator can be directly connected only to a single PE in the network; this implies that the other PEs can communicate with such an external device only through the PE to which the device is connected.

*b. Structure of the Software Process*

The process consists of the avionics programs, the executive software, and the supporting data base. The avionics software is structured along the lines of avionics functions (such as navigation, weapon control, etc.) into a set of programs which are semi-autonomous with regard to hardware resource allocation at process construction time and to scheduling at execution time. Each program in the process is characterized by its expected iteration rate, the I/O messages that it consumes or produces, and the rules to be used to schedule it. Each program can be visualized as a network of tasks, where the I/O relations between individual tasks determine their precedence-successor constraints.

The structure of each program in the process model is defined by the following two related graphs whose nodes represent individual tasks (functional segments) of the program:

The program graph defines all potentially possible execution paths through the task system constituting a program.

The precedence-successor graph defines all precedence-successor constraints in the program due to the I/O relations between the individual tasks; it also shows parallel path substructures of the fork and join type; all constituent tasks of such a parallel substructure must be executed before the paths join together.

One of the initial tasks in the process construction procedure, to be defined later in the sequel, is to construct a maximally-parallel, determinate precedence-successor graph for each program in the process model. The program graph can then be easily derived from such a precedence-successor graph.

The following assumptions have been made about the structure of each program:

It has precisely one entry node (task)

It has precisely one exit node (task)

There are no closed loop cycles through nodes, i.e., all program loops must be embedded within individual nodes (tasks).

A program satisfying these conditions is said to be structurally correct. One of the functions of process construction is to validate assertions about the structural correctness of programs.

Any program eventually must be mapped into, or reside in, a single processing element. If a program exceeds the available capacity of a processing element, it must be split into several programs at process construction time. At the start of process construction, the process is divided into programs along the natural functional boundaries of avionics functions, such that each (sub)function is represented by precisely one program. On completion of process construction, each function is represented by one or several programs, depending on whether the representing program had or had not been split, respectively.

### c. *Process Dynamics at Execution Time*

The execution-time communication between individual process modules and between the process and the external world occurs through the flow and exchange of I/O messages. Not all I/O messages generate bus traffic; for example, some messages may be exchanged between the process modules which reside in the same PEs. Messages which flow between a PE and an external device directly connected to it also do not generate any bus traffic. All I/O messages can be divided into three types: external I/O, interprogram I/O, and intraprogram-intertask I/O messages.

It is assumed that each external I/O device (sensor or actuator) interfaces with the process through an I/O handling program, called the External I/O Handler. An External I/O Handler must always be placed in the same PE to which the external I/O device is connected. To the mission-oriented programs of the process, an External I/O Handler looks like any other program: it may produce outputs, which actually come from a sensor and which may be consumed by other programs, or vice versa. Thus, if a program has not been placed in the same PE with a companion External I/O Handler which is transmitting external inputs to the program, the resulting information flow will cause bus traffic. (A similar statement can be made about external outputs.) One of the functions of process construction is to minimize the potential bus traffic by collocating programs with their External I/O Handlers in the same Processing Elements, especially those that intercommunicate heavily at execution time. The interprogram I/O results in the bus traffic only if the intercommunicating programs are located in different PEs. Thus, the total bus traffic can be decreased by trying to locate the most heavily intercommunicating programs in the same PEs. The intraprogram-intertask I/O does not generate any bus traffic at execution time, but it has the potential to generate bus traffic in the event that the program is further decomposed into several programs. Therefore, the intraprogram-intertask I/O is a factor to be watched when a program is split into several programs at process construction time.

## 2. Interaction With Other Phases of Process Development and With Other Systems

The process construction procedure of Basic Process Constructor (BPC) interacts with the following aspects or phases of process development:

Logical/functional design of the process - for decomposition of the process into major functional components and description of the process interfaces with the external world (sensors and actuators)

Compilation outputs–by using compilation outputs as input information for construction of the computer-independent program models

Process development simulations–by producing the process construction outputs which later are to be used as inputs for system network or functional simulation

Functional model of the real-time executive–by producing the process construction outputs which constitute the control data used by the real-time executive for process initialization and simulated real-time operations

Functional model of bus operations–to produce the control data used to simulate the bus traffic

Functional model of DP/M hardware–to obtain running time estimates.

Hence, the BPC must be designed to interface properly with the following process development subsystems:

With the language which is to be used for processing the process software, and especially with the compilation outputs

With system network and functional simulators

With process (or process model) loaders.

In addition, the programs of the BPC must interface with the Process Development Information System for handling the process construction inputs, outputs, and intermediate results.

### 3. Programming Languages

This subsection states the requirements for the programming languages associated with the development of DP/M processes. Potentially, two programming languages may be involved: that used for coding the process software (Process Development Language) and that used for coding the process development support software, including the BPC (Support Programming Language). Although the long range goal is to have a single high-order programming language for all purposes (i.e., for process as well as for support software), such a goal probably is unrealistic for the initial time period during which the BPC is to be used. Therefore, programming requirements for the two cases have been stated separately.

#### *a. Programming Language for Support Software*

A single programming language which possesses the characteristics listed below is considered sufficient for coding all support software, i.e., the Basic Process Constructor, all process simulators and process development utility routines. This language, to be called the Support Programming Language (SPL), shall

1. Be a scientific programming language with general capabilities similar to those of FORTRAN
2. Have the facility for allocating/deallocating pointer-addressable storage blocks for program variables at execution time (this feature is similar to the controlled storage with BASE attribute in PL/I)

3. Have list processing operators (such as CREATE or DEFINE a list, ADD an element to a list, DELETE a specified element from a list, FIND an element in a list) for linked lists

4. Have character string operators for concatenating two strings, finding a substring in a string, copying a substring from a string, and making string substitutions (assignment statement).

Another feature which is highly desirable but which will not be put as a requirement is the capability to have the data of record type and to refer directly to the individual fields in a record (similar to the PL/I structures or the "data of type record" in PASCAL). Requirements 2, 3, and 4 may be satisfied by implementing the capabilities specified above in the form of special subroutine packages. In such a case, a FORTRAN-like programming language may be used as a basis for the Support Programming Language.

*b. Process Development Language*

Process modules, or models of such modules, will be coded using the Process Development Language (PDL). Programs coded in this language may interface with the process construction procedure of Basic Process Constructor through the process construction input. This interface is considered because certain model data, specified below and needed in process construction, can at least theoretically be generated as a by-product of compilation.

To minimize the manual preparation of process construction inputs, the compiler of the Process Development Language shall be capable of producing the model information described below as a by-product of compiling a process module:

Description of the I/O variables used by the module for intermodule and external communication

Names of the standard (common) subroutines called by the module

Characteristics of the module with regard to its demand on hardware resources (storage, instruction counts, loop statistics, and perhaps, timing)

Information about the module's hierarchical position in the logical structure of the process (i.e., whether it represents a function or a task, and to which part of the process it belongs).

The compiler of PDL shall be capable of producing this information in a form suitable for automatic retrieval from compilation outputs.

Since conventional compilers are not capable of producing all the above-listed information, the following implementational approach is suggested as an interim solution. If certain restrictions on programming conventions are obeyed, then compilation may be able to produce additional information which otherwise would be unavailable. For example, one may require that all I/O variables be specified explicitly in a reserved part of each program; if, in addition, special comment phrases are reserved for describing these variables, then it may be possible to obtain, as a by-product of compilation by a conventional compiler, nearly all information that is needed to describe the I/O variables for process construction. For that purpose, it may also be necessary, or desirable, to develop a special post-compilation text processor whose function would be to extract from the compilation output the data items of interest, and then to edit and store them in a form that can be directly input to the process construction procedure.

## 4. Process Development Information System

The Process Development Information System (PDIS) is a computer based system whose function is to facilitate the management and use of: (1) computer programs associated with process development (there will be two types of such programs—the process development support programs and the programs of the process being developed), and (2) the data used or produced by these programs. In more detail, the functions of PDIS are to

- Increase the effectiveness of process designers and programmers—by minimizing the amount of manual work needed for data preparation and transfer
- Facilitate the management control over process development—by making the process data and software readily accessible
- Facilitate automation of system documentation
- Protect the stored information and control access to it.

The availability of PDIS is considered to be the key factor in automation not only of process construction, but also of other phases of process development, especially of simulations. Therefore, a PDIS shall be developed as a support subsystem for the BPC. The purpose of the present subsection is to state minimal capability requirements for the design of PDIS.

To handle the real-time software development for the avionics processes which are to be actually deployed in the field, advanced versions of the process constructor, possessing a more powerful information system than one specified here in the sequel, shall be needed. This information system will differ from the PDIS of the BPC mainly in the following three aspects:

- It will be accessible through interactive remote terminals
- It will provide the process designer with data and program edit capabilities through interactive remote terminals
- It will be capable of providing extensive services for retrieving information from the process development data base.

Since the cost of including the above three features is relatively high, they have been omitted from the BPC requirements.

### *a. Summary of Functional Capability Requirements and Implementational Considerations*

The PDIS which is to be used to support the BPC shall be capable of

- Making it possible for the BPC programs to communicate through shared data sets/files
- Building up and maintaining model data sets to be used as program inputs
- Storing and saving program outputs for future use
- Retrieving the stored data for examination or editing purposes—by locating, displaying and producing a hard copy (printed report or punched cards) of it
- Controlling the access to the data base in order to protect the stored information
- Providing management control information on the state of the data base.

The Process Development Information System shall contain two types of libraries. The Data Library will handle the model data and simulation outputs, and the Program Library will store the source and object modules of the process model and support (process constructor and simulator) software. The PDIS shall be implemented on a host developmental computer. On that computer system, PDIS shall

Be designed to utilize to the greatest extent all standard processing and utility services provided by that system

Be capable of interfacing with the Process Development Language translator in order to retrieve the source modules for compilation and to store the object modules for future use

Interface with the user through the standard access facilities of the host computer (although the computer access through an interactive remote terminal is very desirable, a terminal equipped with a card reader and a line printer is considered to be adequate for the BPC operations).

The question whether it is better to use a standard Data Management System package, which has approximately the same functional capabilities as specified in the present subsection, or to custom-design and program the required Data Library utility subroutines is not addressed here. This decision is a complex function of the availability and suitability of such a standard package, budget, development schedules, and availability of talent. Therefore, it should be deferred until the actual development of the BPC has been started. Since most computer systems provide reasonable software management facilities, the Program Library shall be developed on the basis of existing facilities.

### *b. Access Control*

Two basic types of access control may be considered:

The access control to protect the integrity of the information stored in PDIS libraries

The access control for screening the users trying to retrieve classified information.

Both types of access control may be needed for the PDIS of an advanced process constructor. Since the applications of BPC will probably not involve the use of any classified information, it is sufficient to provide the BPC only with the access control for protecting the integrity of the Data Library. Protection of software is to be limited to the standard access control facilities provided by the Operating System of the host computer.

In the BPC, the protection of the Data Library (dictionaries as well as data files) shall be accomplished by forcing the user to communicate with the library always through standard PDIS utility subroutines. These subroutines shall be capable of preventing users from writing into or destroying data files that do not belong to them.

### *c. The PDIS Data Library*

The basic function of the PDIS Data Library is to integrate the individual computer programs and procedures (not necessarily computerized), constituting the BPC, into a computerized system by

Facilitating the flow of information through the components of the BPC and, in particular, the sharing of data between such components

Establishing a data interface with the DP/M simulators

Facilitating the input of information into the process construction procedure.

In more advanced versions of Process Constructor, the input interface shall be expanded to include partial outputs of the Process Development Language translator. Since no such special language is absolutely needed and is not foreseen during the initial process construction work, developing an automated input interface may be economically unacceptable, and so the initial inputs to Process Construction Procedure will probably have to be keypunched.

(1) Organization of the PDIS Data Library

The PDIS Data Library shall consist of a D ta Library directory system, the data base, and the Data Library utility subroutines. The data base shall be organized on three hierarchical levels. On the highest level, the entire data base shall consist of one or several, possibly overlapping, user's libraries. Each user's library may contain one or several user's (or logical) files, each such file shall consist of records.

(2) Records

A record may be of variable length. The head portion of the record (record header) shall be of fixed length for all records in the data library and shall contain the record length, the record ID number (key), and the record sequence number for the record group (or type) identified by the record ID number

Sufficient space shall be reserved in the record header for the addition of following optional information

The time and data of record deposition (creation)

A forward pointer to the next record of the same type (key) in the same file

A backward pointer to the last record of the same type (key) in the same file

The first optional item above is especially needed for simulation output, the last two are needed in construction of singly-threaded, doubly-linked lists through the data base.

Each file shall have an initial head record of the same general format as described above, except that key of the record shall be of the special type reserved for file heads only the variable part of that record shall contain the file name/description.

(3) Directory System and Directory Operations

The directory system shall consist of a Master Directory and one or several User's Directories, each identified by the user's name The Master Directory shall contain references to the User's Directories; a User's Directory, to the user's files.

As indicated earlier, file sharing between users shall be allowed. Therefore, a file in the Data Library shall be capable of belonging to one or several user's libraries simultaneously The

file naming system shall make it possible for a user to refer to a file by more than a single name and several users to refer to the same file by different names.

(4) Access Control

The Data Library access control mechanism shall allow two modes of operation, for file modification:

In the read-only access mode, the users are allowed to modify or logically delete (but not purge) their own files only, including those that they share with other users through the directory system

In the unrestricted access mode, the users are allowed to modify, logically delete, or purge (physically delete) any file in the library.

There shall be no access restrictions for reading (copying) of files.

(5) Data Library Utility Subroutines and Their Operations

All user's communication with the Data Library shall be channeled through a standard set of Data Library utility subroutines. These subroutines shall be capable of carrying out the following library use functions/operations:

(1) Data Library Functions/Operations:

Initialize the Data Library and create the Master Directory

Copy the Master Directory into the working area and/or print its contents

Purge all logically deleted users files from the Data Library (a shared file must have been logically deleted by all its former users); if all files of a User's Library have been logically deleted and purged, the User's Directory is purged from the directory system

(2) User's Library Functions/Operations:

Initialize a User's Library and create its directory

Copy a User's Directory into the specified working area and/or print its contents

Add a file to a User's Library by copying it from the working area

Find and copy a file from a User's Library into the working area

Logically delete a single file from a User's Library (i.e., the file may still remain in the libraries of other users)

Logically delete all files from a User's Library and record

(3) File and record functions operations for a file located in a temporary working area:

Initialize a file in a working area

Add a correctly formatted record to the file

Find the next record with a given key in the file (this operation is progressively repeatable)

Delete a record from the file.

The following remarks apply:

Each user's file shall be organized sequentially. It is up to an individual user to establish lists within his file if he needs to do so—the pointer space for that purpose is reserved in the fixed length header portion of each record.

In all above described library functions/operations, it is assumed that all directories affected by a library or file modification are automatically modified subsequent to such a function/operation.

(6) The PDIS Program Library

As already stated, the PDIS Program Library shall be built on the standard software management facilities already available in the computer system on which the Basic Process Constructor is to be implemented.

## 5. Functional Specification of the Process Construction Procedure

### *a. Overview*

This subsection summarizes the functional capability requirements of the DP/M process construction by defining a canonical procedure to be used in the Basic Process Constructor. (In the sequel, this procedure is referred to as process construction procedure.) The Basic Process Constructor essentially is a system of computerized algorithms which interacts with the process designer. The programs constituting this system communicate through common data files which are accessed through the facilities of the PDIS Data Library. The inputs for the Basic Processor Constructor are prepared by the process designer. Two basic input sets are used: one describes the target DP/M computer system for which the process is to be constructed; another describes the process itself. The process is defined by specifying

Its partitioning into avionics subfunctions/functions

The tasks constituting each subfunction in terms of their scheduling rules and their demands on hardware resources

All external, interfunction, and intrafunction-intertask I/O.

As the process construction starts, it is assumed that each subfunction/function is designated to become a DP/M program whose tasks represent its procedures (routines). As the process construction progresses, it may turn out that some programs representing avionics subfunctions/functions do exceed the capacity of a single DP/M Processing Element. Such a program is then split into several separate programs which are to be distributed over several PEs. Thus, by the end of process construction, each avionics subfunction/function has been converted into a single DP/M program or into an integral number of such programs.

### *b. Process Construction Procedure*

The multi-step procedure of the Basic Process Constructor is described next. Although this procedure is to be computerized extensively, the user shall have the option at the end of each step to analyze the process construction data generated up to that point and to overrule certain decisions made by the computerized algorithms. Having this option greatly increases the flexibility of the procedure and is important in the case of hardware resource allocation, for

what is optimal for a particular situation may not precisely fit the generalized optimality models used in process construction algorithms.

Another feature of the procedure is that is that it shall document the process by producing computer-printed reports. These reports shall serve several purposes, the main ones of which are (1) system documentation for configuration and management control and (2) to provide the process designer with information needed in decision making, especially with regard to overruling or influencing the computerized process construction procedures. The final output of the process construction procedure shall contain the model and control data to be used as simulation input. This data shall be organized and formatted so as to be directly inputtable into the appropriate DP/M simulators.

The process construction procedure shall consist of the following nine steps:

Generate computer-independent models of process programs

Analyze and model the process I/O and storage requirements

Generate a computer-dependent process model (i.e., the timing data for each task of every process program)

Analyze process decomposition (i.e., determine the execution paths and validate the structural correctness of every program; construct synthetic models for all programs)

Evaluate process decomposition in order to test whether a process component exceeds the hardware constraints

Synthesize the process (i.e., optimally allocate hardware resources to programs and determine the hardware configuration)

Generate I/O control data

Generate program execution control data

Load all control data, link and load process components (now the process is ready to be executed or simulated)

Figure 171 is a block diagram of the process construction procedure. To simplify the diagram, not all printed reports to be produced are shown in it. Next, each of these steps is described in more detail.

## STEP 1: GENERATION OF A COMPUTER-INDEPENDENT PROCESS MODEL

Process construction starts with the generation of computer-independent models of all process modules. The inputs to the first step may be in two forms:

As an automatically generated by-product of compiling the source programs which constitute avionics functions and tasks

As the data obtained from the manual analysis of avionics algorithms.

A computerized algorithm, called Computer-Independent Model Generator (CIMG), analyzes the input data to do the following:

Define the overall process structure by identifying its programs, including their scheduling rules and iteration rates.

Identify the program inputs and outputs which are due either to the interprogram information flow or to the communication of the process with the outside world (external I/O); identify the storage requirements of the data sets and buffers to be used for storing these inputs and outputs.

Decompose each program into tasks and define the intertask information flow in the form of task I/O lists; identify the storage requirements of the data sets used for interprogram-intertask.

For each task, determine the amount of required computational work in terms of operation counts and loop repetition estimates. (Note that there are alternative techniques for obtaining the running-time estimates. For example, if the task already exists as a subroutine, it might be executed on the target computer or simulated on an instruction-level simulator in order to determine its timing experimentally; another approach, which is recommended but which may be unrealizable for the Basic Process Constructor, is to use a "smart" compiler which would extract model information as a by-product of compilation.)

Determine the total amount of storage required by each task.

The tasks listed above essentially represent transformations which convert and reduce the input information to a form suitable to process construction. For the initial decomposition of the process into programs, it shall be assumed that each avionics subfunction is going to constitute a DP/M program.

After analyzing and reducing the above indicated model data, the CIMG shall

Analyze the intertask information flow in each program to construct (a) the maximally parallel predecessor-successor graph which guarantees program execution determinacy and (b) the related program graph which describes all potentially possible, alternative paths through the program (as noted in Subsection IX.B, these two graphs are related one to another but are not the same)

Compact and write the reduced model data, including both graphs, on the Computer-Independent Model File

Print a report documenting the computer-independent model of the process.

## STEP 2: GENERATION OF A COMPUTER-DEPENDENT PROCESS MODEL

A Basic Process Constructor program, called the Computer-Dependent Model Generator (CDMG), now converts the computer-independent program models, produced in Step 1 and stored on the Computer-Independent Model File, to a computer-dependent form by estimating the task running times and the task memory requirements (the latter is done by considering the word length of the target computer). This program uses two streams of inputs: (1) computer hardware definition and (2) the computer-independent model of the process; it produces the following outputs:

A computer-dependent description of individual tasks, grouped under programs to which these tasks belong, which is written on Computer-Dependent Model File

A printed report separately documenting each task of every program in the process model.

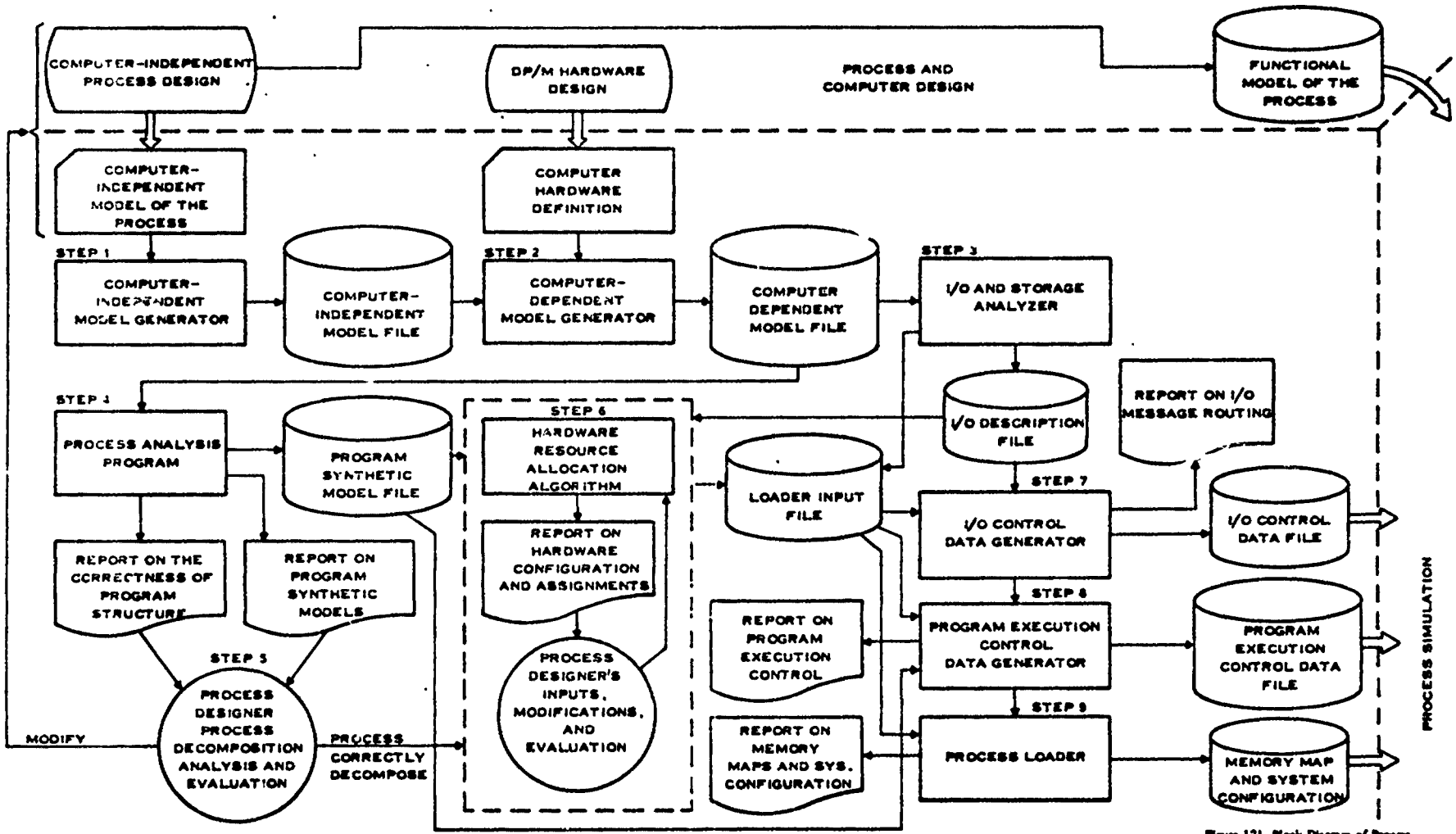
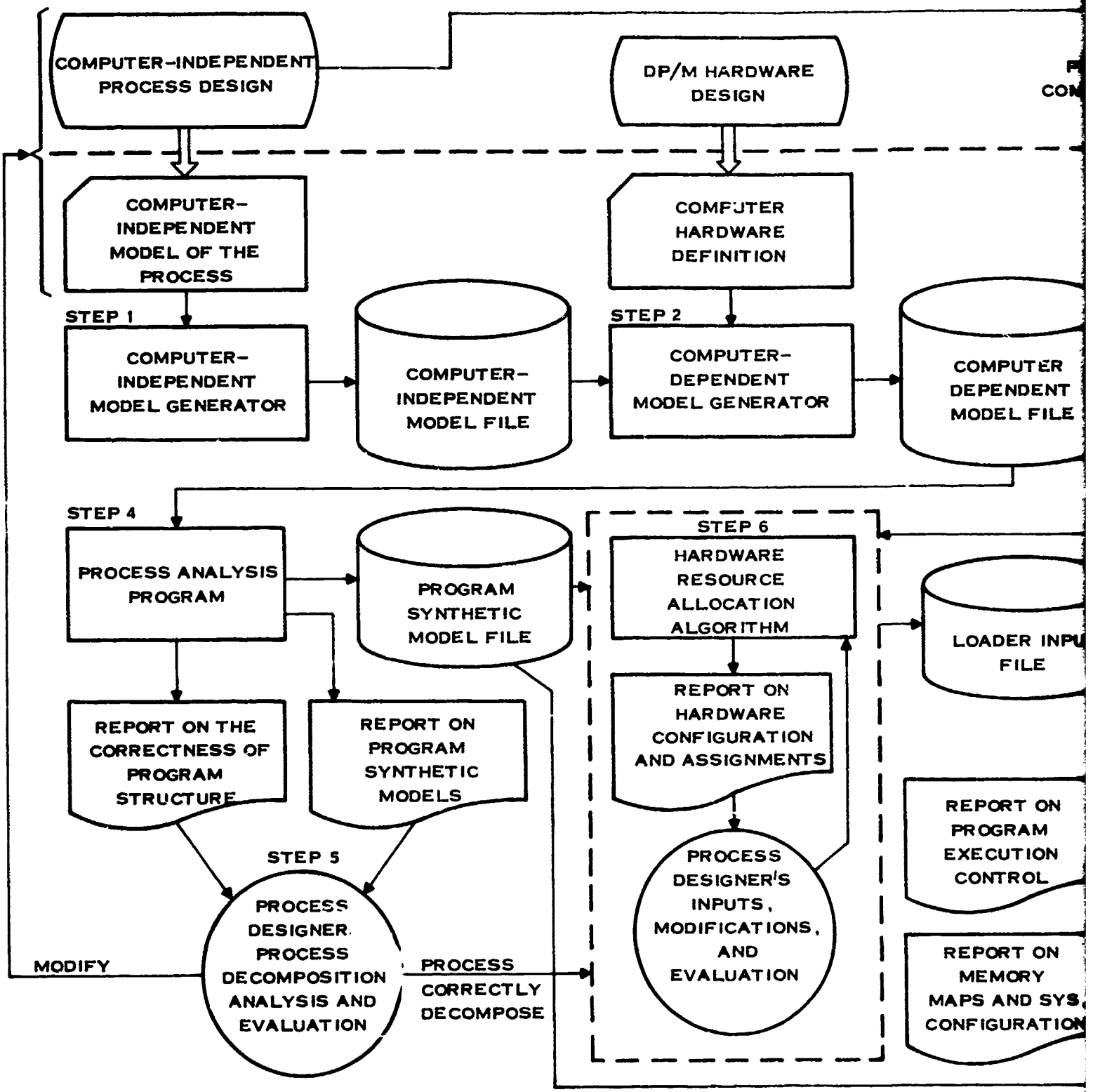



Figure 171. Block Diagram of Process Construction Procedure


COMPUTER-INDEPENDENT PROCESS DESIGN
DP/M HARDWARE DESIGN
COMPUTER-INDEPENDENT MODEL OF THE PROCESS
COMPUTER HARDWARE DEFINITION
STEP 1
COMPUTER-INDEPENDENT MODEL GENERATOR
COMPUTER-INDEPENDENT MODEL FILE
STEP 2
COMPUTER-DEPENDENT MODEL GENERATOR
COMPUTER DEPENDENT MODEL FILE
STEP 4
PROCESS ANALYSIS PROGRAM
PROGRAM SYNTHETIC MODEL FILE
STEP 6
HARDWARE RESOURCE ALLOCATION ALGORITHM
LOADER INPU FILE
REPORT ON THE CORRECTNESS OF PROGRAM STRUCTURE
REPORT ON PROGRAM SYNTHETIC MODELS
REPORT ON HARDWARE CONFIGURATION AND ASSIGNMENTS
REPORT ON PROGRAM EXECUTION CONTROL
STEP 5
PROCESS DESIGNER PROCESS DECOMPOSITION ANALYSIS AND EVALUATION
PROCESS DESIGNER'S INPUTS, MODIFICATIONS, AND EVALUATION
REPORT ON MEMORY MAPS AND SYS CONFIGURATION
MODIFY
PROCESS CORRECTLY DECOMPOSE

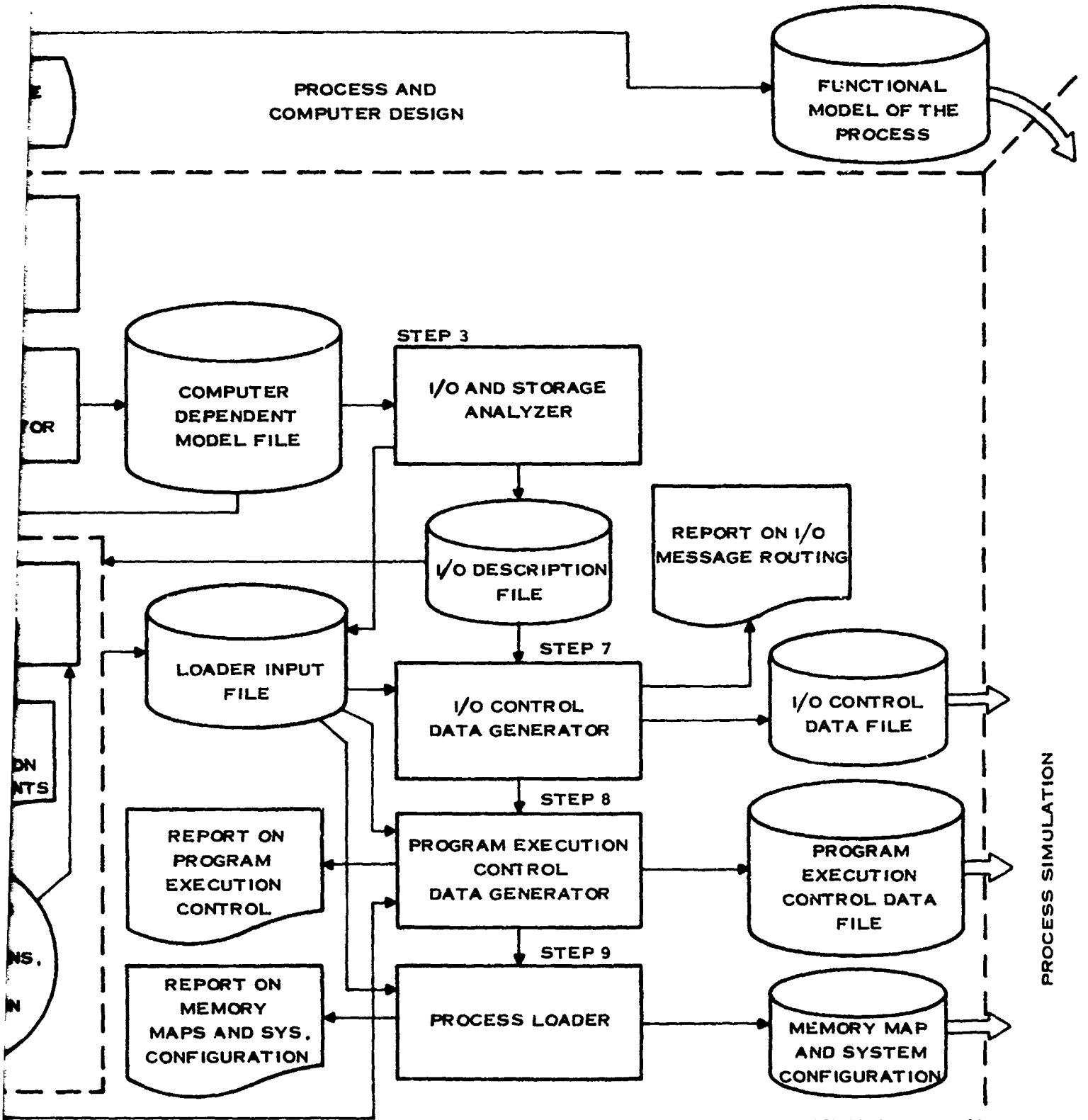



Figure 171. Block Diagram of Process Construction Procedure

## STEP 3: ANALYSIS AND MODELING OF THE PROCESS I/O AND STORAGE REQUIREMENTS

A Basic Process Constructor program, called the I/O and Storage Analyzer (IOSA), analyzes the contents of the Computer-Dependent Model File to retrieve and organize the information to accomplish the program loading/linking and I/O control functions. Since the hardware assignments are not yet known at this point, the information to be produced essentially (a) describes the logical communication between process modules through subprogram calls and I/O messages and (b) summarizes storage requirements for program linking/loading purposes. Specifically, the IOSA produces information on:

- All three types (external, interprogram, and intraprogram) of I/O messages
- The storage space needed for the I/O message data sets and buffers
- Temporary working storage (scratch memory areas) used by individual tasks
- The task permanent memory requirements
- Standard subroutines called by each task—a separate list for each task and the resulting temporary storage requirements per program.

Information on all three types of I/O messages is pooled together for all programs in the process model and written on the I/O Description File, where each I/O message in the model is described in terms of its assigned ID, short descriptor in English, size, production rate, producer, and all consumers. In addition, a communication link matrix describing the interprogram communications and a similar matrix describing the intraprogram-intertask communications, one for each program in the process model, are generated and written on the same file.

Information on the storage requirements for programs and for the individual tasks which constitute these programs shall be stored on the Loader Input File. The following information for each modeled program shall be contained in this file:

- Permanent storage required by each task in the program
- Temporary working storage required by each task in the program
- The list of all standard subroutines called by the tasks of the program and the storage needed for each.

Finally, the I/O and Storage Analyzer prints a report to summarize and document the information stored on both files.

## STEP 4: ANALYSIS OF PROCESS DECOMPOSITION

For each program in the process model, a computerized algorithm of Basic Process Constructor, called the Process Analysis Program (PAP), first analyzes the program graph to do the following: (1) validate the structural correctness of program graph, (2) construct and list all alternative execution paths through the program, (3) using the task timing information from Step 2, compute the expected minimum and maximum path execution times for each path; in addition, if the program graph contains information on branching probabilities (or logical conditions), the PAP computes path probabilities (or logical expressions determining the path selection).

Preceding page blank

Next, the PAP constructs a synthetic model of the program. The following input information is used for that purpose: (1) the computer-independent program model from Step 1, and (2) the list of all alternative execution paths through the program. Using the task timing information from Step 3, the PAP computes the expected minimum and maximum path execution times for each path. If the program graph contains information on branching probabilities or logical conditions, the PAP also computes the path probabilities or logical expressions to model branching during simulation.

The synthetic model shall define a program in terms of (1) program ID, (2) program graph, (3) description and timing of alternative execution paths, (4) storage requirements, (5) message type IDs for all the I/O messages consumed or produced by the program, and (6) expected execution rates and other control information needed by the DP/M executive. In addition, each task of the program shall be described individually as much as the level of modeling requires doing so.

The PAP shall produce the following analysis outputs for each processed program model:

(1) A printed report on the validation of structural correctness of program graph

(2) A printed report on (a) the alternative execution paths through the program graph by specifying each path in terms of an ordered node (task) list, execution probability (or logical conditions), and minimum and maximum run time estimates, (b) the synthetic model of the program, including the estimated demands on hardware resources (timing and storage)

(3) A punched card image version of the synthetic model, which is to be stored in a form directly acceptable by the simulator as Program Synthetic Model File.

## STEP 5: EVALUATION OF PROCESS DECOMPOSITION

In the BPC, this step shall be carried out manually. Using the output information produced in Step 4, the process designer analyzes the model of each program to determine whether the program does not exceed the processing-time and storage limitations of hardware. The required program iteration rates and the portion of real-time that can be allocated to the program are taken into account to decide whether the timing constraints will remain unviolated. If no program violates the hardware limitations, the process is correctly decomposed, and the process construction procedure can continue to Step 6; otherwise, it must return to Step 1, or to that phase of process development which precedes Step 1 of process construction, in order to re-partition into programs an appropriate portion of the process.

The conclusion that the process decomposition is incorrect means that at least one program in the process model exceeds either the real-time processing constraints or the memory capacity of a single DP/M Processing Element. Several alternative options, depending on the cause of the difficulties, are now open to the process designer:

Split each violating program into several new programs, each satisfying all required real-time and hardware constraints; for that purpose, use the maximally parallel predecessor-successor graph of the violating program, obtained in Step 1, to determine the best program partitioning; the optimality criteria that may now be used are, for example, minimization of the number of splits (or of new programs) and/or minimization of the resulting longest execution path; the second criterion implies use of the possible parallelisms indicated in the maximally parallel predecessor-successor graph of the original program;

Modify the logical partitioning of the process into avionics subfunctions, which technically precedes Step 1 of process construction, and decompose each subfunction leading to a violating program into several subfunctions.

The second approach in general will lead to a larger perturbation in the original process partitioning than the first; what is even worse from the managerial viewpoint is that it leads to steps that precede the process construction Therefore, the first approach is recommended; however, if the structure of program does not possess enough potential parallelism, the sec d approach may be unavoidable in the case of timing-constraint violations.

## STEP 6: PROCESS SYNTHESIS: ALLOCATION OF HARDWARE RESOURCES

A computerized Basic Process Constructor algorithm, called Hardware-Resource Allocation Algorithm (HRAA), shall be used to assign hardware resources to the programs in process model and to determine tl best hardware configuration. The synthetic models of process programs, produced in Step 4, and the I/O Description File from Step 3, provide the input information needed by this algorithm.

The hardware resource assignment is to be optimal in the sense that it must try to minimize the potential real-time traffic, or interaction, across the natural boundaries of hardware units. In the present context, the individual Processing Elements of a DP/M system represent such hardware units. (Note that this optimization problem is equivalent to the problem of minimizing the required hardware resources.) The HRAA shall be designed to permit the mapping of a subset of proven programs into the pre-specified PEs of DP/M, with the remaining programs to be mapped optimally. It shall produce the following outputs:

A printed report showing the resulting hardware configuration, the PE assignments to programs, and the projected bus traffic and Processing Element loads

A summary of the hardware configuration and PE assignments to programs which is to be added to the Loader Input File.

The process construction procedure shall permit the process designer to overrule the hardware resource assignments made by the computerized HRAA. To assist the process designer in this task, a printed report (the first item immediately above) shall be provided. After examining the report, the process designer shall be able to choose between the two courses of action: (a) to have further modifications made by changing the hardware configuration or the program-to-PEs assignments and then by executing the HRAA or (b) to proceed to Step 7.

## STEP 7: GENERATION OF I/O CONTROL DATA

A Basic Process Constructor program, called I/O Control Data Generator (IOCDG), combines the results of the hardware resource assignments (made in Step 6 and stored in the Loader Input File) with the model of the I/O interaction between process modules (constructed in Step 3 and stored in the I/O Description File) to generate the control data for handling the process I/O in simulated real-time. For system network or functional simulation, the I/O control data describes each type of I/O message in terms of its ID, size, producer, consumers, scheduling rules, and hardware routing information.

This program shall produce the following outputs:

A printed report describing the routing, timing, and scheduling control of the external, bus (interprocessor), and intraprocessor-interprogram I/O messages

A data file (punched card images), called the I/O Control Data File, containing the description of the process I/O and all additional information that is needed to control the I/O traffic in DP/M simulators (all data on the I/O Control Data File shall be formatted to be directly input into the simulators).

## STEP 8: GENERATION OF PROGRAM EXECUTION CONTROL DATA

A Basic Process Constructor program, called Program Execution Control Data Generator (ECDG), shall pool together and edit all data needed to:

Initialize the process in simulated real-time

Control the simulated execution of programs

The inputs to this step come from:

The program Synthetic Model File

The Loader Input File

The output data shall be organized and formatted to represent the actual tables and lists used by the simulation models of DP/M Executives. The information contained in the output data shall describe each process program by its ID, scheduling and priority rules, program input and output, running time estimates for individual tasks, the maximally parallel predecessor-successor graph, and the hardware assignments made in Step 6. Every task belonging to a program shall be defined, as much as the level of modeling requires doing it, in similar terms. To facilitate simulation, the output shall also describe all alternative execution paths through the program graph, where each path is specified by an ordered list of tasks (nodes in the program graph), task transition probabilities, and task execution times. All outputs shall be stored on the Program Execution Control Data File, the information on which will be organized and structured to be directly usable by the simulators; they shall also appear as a printed report.

## STEP 9: PROCESS LINKING/LOADING AND GENERATION OF A SYSTEM CONFIGURATION DOCUMENT

In construction of an actual process for the target DP/M system, the first function of this step represents linking and loading of the process data and software. For that purpose, one would use a special linking loader. In construction of process models for simulation, this step reduces itself to the allocation of simulated storage and generation of memory maps. A Basic Process Constructor program, called Process Loader, shall use the information on storage requirements, stored on the Loader Input File, to construct memory maps for each Processing Element. These maps serve a two-fold purpose:

They allow to study in detail certain hardware and software design problems

They are actually needed for functional (but not for system network) simulation.

The second function of this step is to produce a system configuration document. This document shall consist of two master directories. The first directory, to be called the Software Module Master Directory, shall list and describe all software modules in the process which are to be loaded independently (i.e., those which are considered to be external modules with regard to program loading functions). The other directory, to be called Data Set Master Directory, shall describe all defined data sets (i.e., arrays, buffer events, file space) for which the memory is to be allocated independently. The entries in both directories shall indicate the PE assignments

made in Step 6. The outputs of the Process Loader shall be written into the Process Memory Map and System Configuration File; they shall also appear in the form of a printed report.

## 6. Process Construction Algorithms and Input/Output Data Sets

The purpose of this subsection is to specify the key algorithms and major input/output data sets (files) for the Basic Process Constructor. The key algorithms are specified by associating the steps of the process construction procedure, outlined in Subsection IX.C.5, with various algorithms discussed in Subsection IX.B.5. To avoid repetition these algorithms are not completely restated in the sequel. Algorithms of only two types are specified; those to be used for process analysis, and those for process design. As pointed out in the beginning of Subsection IX.B.5, the algorithms which have not been specified in this document are data transformation algorithms: They are not only relatively simple but also depend on the detailed design of the data sets of the BPC. After the detailed design of the data sets becomes available, these algorithms can be designed in a straight-forward fashion from their high-level, functional descriptions given in Subsection IX.B.5, and from the data set specifications.

In order not to restrict the software design and implementation alternatives for the BPC, only the exogenous input, endogenous, and certain intermediate (i.e., those to be used to interface with the process designer) output data sets of the process construction procedure are specified here; the others can be designed in many different ways by using the functional overview given in Subsection IX.C.5 as a guideline.

### *a. Process Construction Algorithms*

#### STEP 1: GENERATION OF A COMPUTER-INDEPENDENT PROCESS MODEL

Two process analysis algorithms are used in this step:

An algorithm which for each program in the process model generates the maximally-parallel predecessor-successor graph with the following two properties:

(a) The graph guarantees program execution determinacy

(b) It is of minimum complexity (has the least number of links) among all maximally-parallel determinate graphs.

In the present context, the program execution determinacy means that the actual order of execution of the tasks which belong to a group of conjunctive parallel paths does not affect the results of computation.

Such a graph can be constructed by analyzing the input/output relationships between the tasks of a graph (these tasks constitute the nodes of the graph). A technique for constructing the maximally parallel predecessor-successor graph of a program is outlined in reference [11].

An algorithm for constructing the program graph, which shows all alternative (i.e., disjunctive) execution paths through the program; once the maximally parallel predecessor-successor graph is available, the program graph can be derived from the former by stringing out the paths of every conjunctive group of parallel paths in an arbitrary sequential order.

STEP 2: GENERATION OF A COMPUTER DEPENDENT PROCESS MODEL

All algorithms are data-transformation and reduction procedures

STEP 3: ANALYSIS AND MODELING OF THE PROCESS I/O AND STORAGE REQUIREMENTS

All algorithms are data-transformation and reduction procedures.

STEP 4: ANALYSIS OF PROCESS DECOMPOSITION

Algorithms for the following analysis functions are required:

Construction of the Transition Probability Matrix, T, (or of a similar matrix which would specify the logical branching conditions in selection of alternative program execution paths).

Construction of Node Adjacency matrix from the program graph

Topological ordering of the nodes in the program graph

Generation of Path Matrix

Generation (enumeration) of all feasible execution paths by listing in proper order the nodes of each path

Computation of the path timing estimates and probabilities

Computation of memory requirements.

The topological sort based on Fulkerson's Rule, which has been referred to as Algorithm 1 in Subsection IX.B.5 shall be used to accomplish the third function above

For the fourth function (i.e., construction of a Path Matrix), Warchall's Algorithm, known as Algorithm 2 in Subsection IX.B.5, shall be used.

Algorithm 3 of Subsection IX.B.5 shall be used to generate all alternative execution paths through the program.

All other functions in this step constitute simple data analysis reduction procedures

STEP 5: EVALUATION OF PROCESS DECOMPOSITION

It is a human decision task The decision whether to consider the process as being correctly decomposed or not is to be made on the basis of the information contained in two printed reports produced in Step 4, one on the correctness of program structure and the other, on program synthetic models.

STEP 6: ALLOCATION OF HARDWARE RESOURCES

This step is a two-stage iterative procedure in which the process designer is in a loop with a computerized (sub)optimal hardware resource allocation algorithm As stated previously, the two main functions of this algorithm are

To assign PEs to the programs constituting a process such that:

All real-time and storage constraints are satisfied

The total bus traffic due to the interprogram communications is minimized

To derive the corresponding hardware configuration requirements which would specify:

The total number of PEs needed

All PEs which are to be connected to external sources or sinks and into which appropriate External I/O Handlers (programs) are to be placed.

As stated in Subsection IX.B.5, four candidate algorithms have been considered for (sub)optimal hardware resource allocation. The results of the investigation of these algorithms indicate that either Algorithm 4A (the Branch and Bound method of linear integer programming) or Algorithm 4B (MacQueen's K-Means Method) should be the final choice. Algorithm 4A should be selected if the optimality of allocation, but not its computational cost, is the primary consideration; Algorithm 4B should be selected if the cost of computation and limitations of the developmental computer are important.

STEPS 7 (GENERATION OF I/O CONTROL DATA), 8 (GENERATION OF PROGRAM EXECUTION CONTROL DATA), and 9 (PROCESS LINKING AND LOADING)

These steps involve only data transformation and reduction procedures. The main function of the procedure constituting Step 9 is storage allocation for every PE in the determined DPM configuration. Storage is to be allocated for both the programs and data sets buffers according to the hardware allocations made in Step 6.

*b. Data Sets of the Basic Process Constructor*

The following files reports of the Basic Process Constructor have been identified and are described in Appendix A

Input data sets and input documentation reports

Computer-Independent Model of the Process (Input File)

Report on Computer-Independent Model of the Process

Computer Hardware Definition (Input File)

Report on Computer Hardware Definition

Process construction endogenous outputs

I/O Control Data File

Report on I/O Message Routing

Program Execution Control Data File

Report on Program Execution Control

Memory Map and System Configuration File

Report on Memory Maps and System Configuration

Intermediate output reports

Report on the Correctness of Program Structure

Report on Program Synthetic Models

Report on Hardware Configuration and Assignments.

The output reports listed above under the third category are those intermediate reports by means of which the designer interfaces with the process construction procedure. Their main function is to keep the designer informed about the progress and partial results of process construction at two critical stages of the process construction procedures: the process decomposition analysis (Steps 4 and 5) and the hardware resource allocation (Step 6).

## 7. Implementational Requirements

This subsection summarizes the implementational requirements of the Basic Process Constructor. Since there is no intent at the present time to foreclose on any of the significant design and implementation alternatives or to dictate the selection of the developmental host computer, the implementational requirements have been reduced to a summary of the projected minimal processing capability requirements for the developmental host computer system. These capability projections have been derived from the assumption that the same host computer will be used for the following functions:

Process construction under the Basic Process Constructor

All basic types of process and hardware simulations (system network, functional, instruction level)

Development of the support software

Process Development Information System operations

Compilation of DP/M programs to be simulated by the instruction level simulator or executed on a DP/M system.

### *a. Software and Support Function Requirements*

The developmental host computer shall be provided with and/or shall be capable of supporting.

The high level programming language(s) in which the Basic Process Constructor, simulation software, the utility subroutines of Process Development Information System, and other support programs are to be coded

Compiler for the high-level programming language of the DP/M system

Software facilities needed to accomplish the Program Library functions of the Process Development Information System (these functions comprise the standard operations used in handling the partitioned source and object code libraries).

If there is no intent to use the host computer of the Basic Process Constructor for the development of software for DP/M systems, the second requirement listed above may be relinquished.

### *b. Hardware Requirements*

The following features/characteristics of the host computer hardware represent the projected minimal hardware capability requirements:

At least an equivalent of 100K 32-bit words of central memory available to the user

At least 10 million 32-bit words worth of disk storage

At least one 80-column (or wider) line printer

At least one card reader

At least one card punch

At least one magnetic tape drive.

## D. PROCESS CONSTRUCTION CASE STUDY

To evaluate the fundamental steps involved in the basic process construction methodology, a case study involving computerized and manual procedures was performed. The effort involved the allocation of a network of DP/M PEs to a set of avionic processing functions for a hypothetical mission. The mission and functions selected were similar to a Close Air Support mission and the processing functions were those associated with a NARBS (Night Angle Rate Bombing) mode. Three sets of data were formed. An existing data base of avionics functions that was described in terms of basic operations (i.e., number of additions, subtractions, multiplies, divides, logical operations, sines, cosines, etc.) was used as the computer-independent description of each of the processing algorithms. A model description of the DP/M PE computational characteristics (i.e., operation execution times and memory allocations) was defined. A third data set was made up of the input/output variables associated with the avionic functions. The I/O data served as a basis for the generation of three reports. The most fundamental I/O data report was an alphabetized listing of all signals in the system. The second report showed each I/O signal and the consumers (avionic programs) of the signal. The final report listed each avionics program and its input and output variables. The program input variables were grouped according to the source of the signals to give a first cut indication of likely collections of variables for bus messages.

The avionic program (or subfunction) input/output data was combined with the computer algorithm independent operation summary, and this information was used by a series of computer programs to automatically calculate the execution time and memory allocation requirements for the avionics subfunction with respect to the DP/M PE model. An output report was generated, summarizing the memory, timing, and input/output requirements for each subfunction, plus a set of summary records (punched cards). These summary records were processed by a task analysis program to verify the structural corrections of the avionic subfunction directed graph, to analyze the number of transitional execution paths through the program based on assigned probability, and to give an estimate of processor execution time loading for the given avionics subfunction. This information was used for the manual resource allocation of avionic functions to DP/M PEs.

Following the manual resource allocation process, a system network simulation was made of this hypothetical mission and performance reports gathered for a sample time interval.

## 1. Mission Segment Description Summary

The chosen CAS NARBS mode mission segment was made up of a number of active processing programs that are summarized along with their respective iteration rates in Table 34.

**TABLE 34. CLOSE AIR SUPPORT MISSION/ATTACK SEGMENT–NARBS MODE PROGRAM SUMMARY**

| Program | Iteration Rate/Sec |
|---|---|
| AIRDATA/AIRDATASS | 16 |
| CONPANSS | 16 |
| CNPANMAN | 1 |
| DISPLMAN | 4 |
| DISPLASS | 4 |
| FLTCONT/FLTCONSS | 64 |
| MISSMGT | 1 |
| NAVFILT | 1/8 |
| RADARSS | 32 |
| RADALTSS | 32 |
| ECMSS | 32 |
| COMMSS | 16 |
| AIRCRFSS | 2 |
| GYRACESS | 32 |
| STOMANSS | 32 |
| INSNAV/INSSS | 32 |
| DOPPLER/DOPPSS | 32 |
| LORAN/LORANSS | 32 |
| RHAWGTL | 32 |
| FLIRPTG/FLIRSS | 32 |
| NARBSWD | 32 |

## 2. Mission Software Models

Each of the active mission processing programs was described in terms of a directed graph along with each node (or task) within the graph. A summary of this description is shown by the reports given in Figures 172 and 173. This generic description of the computer operations for each program task was derived from the math and logic algorithms used for each avionics function. A summary of the usage of these generic operations per application program is given in Table 35. Also included in the description of each program node is its iteration rate, subprograms called, input/output variables, and a list of candidate successor tasks and the transitional probability for each successor task.

```
**********                                                    01/08/75
                    SYNTHETIC MODEL OF A PROGRAM

PROGRAM NAME                                                  AIRDATA
PROGRAM ID NUMBER                                                   1

FOR COMPUTER WHOSE ID IS                                    DP/M   PE

NAME OF THE INITIAL PROCEDURE                                   PTCN
NUMBER OF PROCEDURES                                               4

PERIODICITY (=NUMBER OF REPETITIONS PER MAJOR CYCLE)              16

NUMBER OF INPUTS                                                   9
NUMBER OF OUTPUTS                                                 12
**********
```

Figure 172. Program Synthetic Model Summary Report

### 3. Computer Definition Model

In conjunction with the computer-independent model descriptions of the avionic processing functions, a computer-dependent model of the DP/M Processing Element was defined. This PE model description was based upon conservative estimates of instruction execution times. Math subroutine execution times are representative of those functions on current airborne minicomputers. A summary of the DP/M PE hardware model used in the case study is given in Figure 174.

### 4. Application Program Timing and Sizing Effort

A task analysis program was used to process the avionic program description and PE computer model description and to produce a series of reports summarizing program execution paths, processor loading and memory requirements for the program. Figure 175 shows the report generated for one of the case study avionic programs whose program graph is shown in Figure 176.

With the given avionic program description and computer model used for this sizing effort, the following observations were made:

(1) The processor loading for each application program was such that it would have no difficulty being processed in only one PE.

(2) Memory resources requirements of each application program were small enough to fit into a 4K memory in the majority of cases.

```
**********************************************************************
                                                          01/08/75
                  SYNTHETIC MODEL OF A PROCEDURE

PROCEDURE NAME                                                  PTCN
          ID NO.                                                 101
          COUNT FOR PROGRAM  AIRDATA                               1
----------------------------------------------------------------------
PERMANENT MEMORY (16-BIT WORDS)                                  505
TEMPORARY MEMORY (16-BIT WORDS)                                   41

REPETITIONS (LOOPS)
          MINIMUM NUMBER                                           1
          MAXIMUM NUMBER                                           1

RUNNING TIME PER REPETITION (MICROSECONDS)
          MINIMUM                                          2547.500
          MAXIMUM                                          2571.500


NUMBER OF SUBROUTINE PROCEDURES CALLED                             0

NUMBER OF INPUT VARIABLES USED                                     9
NUMBER OF OUTPUT VARIABLES PRODUCED                               12


NUMBER OF PROCEDURE SUCCESSORS                                     1
----------------------------------------------------------------------
PROCESSING LOAD PER REPETITITON (ITERATION)

SINGLE PRECISION ARITHMETIC OPERATIONS
          ADDITIONS/SUBTRACTIONS                                  12
          MULTIPLICATIONS                                         23
          DIVISIONS                                                6

DOUBLE PRECISION ARITHMETIC OPERATIONS
          ADDITIONS/SUBTRACTIONS                                   0
          MULTIPLICATIONS                                          0
          DIVISIONS                                                0

LOGICAL/BOOLEAN OPERATIONS                                         0

AVG. NO. OF CONTROL OPERNS PER ONE ARITH OPERN                   7.0
AVG. NO. OF CONTROL OPERNS PER ONE LOGIC OPERN                   3.0

SINGLE PRECISION STANDARD SUBROUTINE CALLS
          SQUARE ROOT                                              5
          SINE/COSINE                                              0
          ARCTAN                                                   0
          EXPONENTIAL                                              0
          LOGARITHM                                                4
          ARCSIN/ARCCOS                                            0

DOUBLE PRECISION STANDARD SUBROUTINE CALLS
          SQUARE ROOT                                              0
          SINE/COSINE                                              0
          ARCTAN                                                   0
          EXPONENTIAL                                              0
          LOGARITHM                                                0
          ARCSIN/ARCCOS                                            0
**********************************************************************
```

Figure 173. Task Synthetic Model Summary Report (Sheet 1 of 2)

LIST OF INPUT VARIABLES REQUIRED
BY PROCEDURE PTCN

| TYPE | VAR ID | VARIABLE NAME | VARIABLE ABREV'N | SOURCE | ITER RATE | BITS |
|---|---|---|---|---|---|---|
| P | 1 | IND TEMP | TI | ADSENS | 2 | 12 |
| P | 1 | IND ANG ATTK | AOAI | ADSENS | 8 | 12 |
| P | 1 | IND STAT PRES | PSI | ADSENS | 16 | 16 |
| P | 1 | IND TOT PRES | PTI | ADSENS | 16 | 16 |
| P | 1 | TOT PRES STATUS | S101 | AIRDATSS | 1 | 1 |
| P | 1 | STAT PRES STATUS | S102 | AIRDATSS | 1 | 1 |
| P | 1 | TEMP STATUS | S103 | AIRDATSS | 1 | 1 |
| P | 1 | AOA STATUS | S104 | AIRDATSS | 1 | 1 |
| P | 1 | SCALES UP STATUS | S2901 | CONPANSS | 16 | 1 |

LIST OF OUTPUT VARIABLES PRODUCED
BY PROCEDURE PTCN

| TYPE | VAR ID | VARIABLE NAME | VARIABLE ABREV'N | ITER RATE | BITS |
|---|---|---|---|---|---|
| P | 1 | HUD SCL VTAS | C101 | 16 | 10 |
| P | 1 | HUD SCL ALT | C102 | 16 | 10 |
| P | 1 | HUD/BASE/VEL | Y101 | 16 | 10 |
| P | 1 | HUD/BASE/ALT | Y102 | 16 | 10 |
| P | 1 | HUD/THERM SCL LT | Y103 | 16 | 10 |
| P | 1 | HUD/THERM SCL RT | Y104 | 16 | 10 |
| P | 1 | ANG OF ATTK | AOA | 16 | 16 |
| P | 1 | SIDESLIP ANGLE | BETA | 16 | 16 |
| P | 1 | BARO ALT | HB | 16 | 16 |
| P | 1 | AIR DENSITY | RHO | 16 | 16 |
| P | 1 | IND AIRSPEED | VC | 16 | 16 |
| P | 1 | TRUE AIRSPEED | VTAS | 16 | 16 |

SUCCESSORS OF PROCEDURE PTCN

| SUCCESSOR NO. | NAME | ID NO. | TRANS.PROB. |
|---|---|---|---|
| 1 | AOA | 102 | 1.000 |

Figure 173. Task Synthetic Model Summary Report (Sheet 2 of 2)

## TABLE 35. GENERIC PROCESSING OPERATIONS AND INSTRUCTION COUNTS SUMMARY

| Program | ADD/SUB | MULT | DIV | LOGIC | SQRT | SIN/COS | ARC SIN/ COS | ARC TAN | EXP | LOGN |
|---|---|---|---|---|---|---|---|---|---|---|
| AIRDATA/AIRDATASS | 18 | 31 | 6 | 332 | 5 | 0 | 0 | 0 | 0 | 4 |
| CNPANSS | 0 | 0 | 0 | 120 | 0 | 0 | 0 | 0 | 0 | 0 |
| CNPANMAN | 0 | 0 | 0 | 75 | 0 | 0 | 0 | 0 | 0 | 0 |
| DISPLMAN | 0 | 0 | 0 | 350 | 0 | 0 | 0 | 0 | 0 | 0 |
| DISPLASS | | | | | | | | | | |
| FLTCONT/FLTCONSS | 5 | 4 | 0 | 377 | 0 | 0 | 0 | 0 | 0 | 0 |
| MISSMGT | | | | | | | | | | |
| NAVFILT | 2760 | 3810 | 210 | 0 | 110 | 0 | 0 | 0 | 0 | 0 |
| RADARSS | 34 | 3 | 4 | 389 | 0 | 1 | 2 | 0 | 0 | 0 |
| RADALTSS | 0 | 0 | 0 | 64 | 0 | 0 | 0 | 0 | 0 | 0 |
| ECMSS | 0 | 0 | 0 | 80 | 0 | 0 | 0 | 0 | 0 | 0 |
| COMMSS | 0 | 0 | 0 | 93 | 0 | 0 | 0 | 0 | 0 | 0 |
| AIRCRFSS | 0 | 0 | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 |
| GYRACESS | 0 | 0 | 0 | 19 | 0 | 0 | 0 | 0 | 0 | 0 |
| STOMANSS | 0 | 0 | 0 | 5880 | 0 | 0 | 0 | 0 | 0 | 0 |
| INSNAV/INSSS | 72 | 75 | 1 | 865 | 20 | 7 | 0 | 3 | 0 | 0 |
| DOPPLER/DOPPSS | 6 | 8 | 1 | 159 | 2 | 4 | 0 | 0 | 0 | 0 |
| LORAN/LORANSS | 50 | 78 | 8 | 778 | 0 | 23 | 0 | 3 | 0 | 0 |
| RHAWGTL | 12 | 28 | 0 | 205 | 0 | 8 | 0 | 0 | 0 | 0 |
| FLIRPTG/FLIRSS | 60 | 40 | 8 | 705 | 2 | 6 | 0 | 3 | 0 | 0 |
| NARBSWD | 74 | 118 | 25 | 1159 | 5 | 34 | 1 | 3 | 0 | 0 |

```
**********                                                    **********
                                                              01/08/75
                      COMPUTER DEFINITION

COMPUTER ID NAME                                              DP/M  PE
        ADDITIONAL IDENTIFICATION                             PROTOTYP

ALL TIMING DATA IS IN MICROSECONDS
-------------------------------------------------------------------------

MEMORY SIZE (16-BIT WORDS)                                        4096

SPEED OF SINGLE PRECISION OPERATIONS
           ADDITION/SUBTRACTION  -  MIN T                        2.000
                                 -  MAX T                        4.000
           MULTIPLICATION        -  MIN T                       20.000
                                 -  MAX T                       20.000
           DIVISION              -  MIN T                       20.000
                                 -  MAX T                       20.000

SPEED OF DOUBLE PRECISION OPERATIONS
           ADDITION/SUBTRACTION  -  MIN T                       60.000
                                 -  MAX T                       70.000
           MULTIPLICATION        -  MIN T                      250.000
                                 -  MAX T                      350.000
           DIVISION              -  MIN T                      250.000
                                 -  MAX T                      350.000

LOGICAL/BOOLEAN OPERNS-  AVG T                                    3.000
CONTROL OPERNS         -  AVG T                                   2.500
SUBROUTINE LINKAGE     -  AVG T                                   4.000

AVG EXEC T FOR STANDARD SINGLE PRECISION SUBROUTINES
           SQUARE ROOT                                          170.000
           SINE/COSINE                                          115.000
           ARCTAN                                               135.000
           EXPONENTIAL                                          215.000
           LOGARITHM                                             85.000
           ARCSIN/ARCCOS                                        300.000

AVG EXEC T FOR STANDARD DOUBLE PRECISION SUBROUTINES
           SQUARE ROOT                                         4900.000
           SINE/COSINE                                         1100.000
           ARCTAN                                              1200.000
           EXPONENTIAL                                         1500.000
           LOGARITHM                                           1200.000
           ARCSIN/ARCCOS                                       6000.000
**********                                                    **********

MEMORY ALLOCATION GUIDELINES

SINGLE PREC ARITH OPER'NS - PERM MEM                              1.5
                          - TEMP MEM                              0.5
DOUBLE PREC ARITH OPER'NS - PERM MEM                              2.0
                          - TEMP MEM                              1.0
LOG'L,CONTROL,SUB LINK    - PERM MEM                              1.5
                          - TEMP MEM                              0.0
```

Figure 174. Computer Definition Model

DP/M PROCESS CONSTRUCTION DATE: 750116

PROGRAM MODEL REPORT - PAGE: 1

PROGRAM NAME: INSNAV
ID #: 0
# OF ITERATIONS PER SECOND: 32

# OF NODES: 10
INPUTTED ID # OF ENTRY NODE: 106

COMPUTER MODEL: DP/M PROTOTYPE

| NODE ID# | NODE NAME | TOPO ID# | PERM MEM. | TEMP MEM. | MIN REP# | MAX REP# | MIN RUN T(USECS) | MAX RUN T(USECS) | IN-DEGR | SUCCESSOR ID# | T.PROB |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 106 | IENT | 1 | 96 | 38 | 1 | 1 | 168 | 168 | 0 | 107 | 0.0100 |
| | | | | | | | | | | 108 | 0.9800 |
| | | | | | | | | | | 114 | 0.0100 |
| 107 | IINI | 2 | 145 | 5 | 1 | 1 | 1508 | 1512 | 1 | 108 | 1.0000 |
| 114 | IAMS | 3 | 36 | 0 | 1 | 1 | 63 | 63 | 1 | 115 | 1.0000 |
| 108 | IPLT | 4 | 1273 | 53 | 1 | 1 | 3356 | 3438 | 2 | 109 | 0.3000 |
| | | | | | | | | | | 110 | 0.7000 |
| 109 | INFC | 5 | 204 | 8 | 1 | 1 | 332 | 366 | 1 | 110 | 1.0000 |
| 110 | INOT | 6 | 120 | 0 | 5 | 5 | 210 | 210 | 2 | 111 | 1.0000 |
| 111 | INAS | 7 | 72 | 0 | 1 | 1 | 126 | 126 | 1 | 112 | 0.9800 |
| | | | | | | | | | | 113 | 0.0100 |
| | | | | | | | | | | 115 | 0.0100 |
| 112 | IWND | 8 | 48 | 2 | 1 | 1 | 114 | 118 | 1 | 113 | 0.9900 |
| | | | | | | | | | | 115 | 0.0100 |
| 113 | IHUD | 9 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 115 | 1.0000 |
| 115 | IEND | 10 | 0 | 0 | 1 | 1 | 0 | 0 | 4 | | |

Figure 175. INSNAV Sizing Report (Sheet 1 of 2)

(3) Allocation of programs from a resource standpoint would have to be made based on memory required, rather than processor loading.

It should be noted that these observations were applicable for the description data base of programs used for the case study, and such observations are not intended to be universal in nature.

## 5. Partitioning Results

This subsection describes the results of the nominal allocation of application programs to DP/M resources based on the output of the Process Constructor and the location of the sensor/actuator (if any) associated with each application program. A system-level flow of control and messages was constructed and is shown in Figure 177. The allocation process was manual

```
**********************************************************************

                    DP/M PROCESS CONSTRUCTION              DATE: 750116
                    -------------------------

                PROGRAM MODEL REPORT - PAGE:  2

           PROGRAM NAME:                                        INSNAV
                   ID #:                                             0
           # OF ITERATIONS PER SECOND:                              32

           # OF NODES:                                             10
           INPUTTED ID # OF ENTRY NODE:                           106

           COMPUTER MODEL:                                DP/M PROTOTYPE
**********************************************************************

# OF EXECUTION PATHS =                                             17

PATH
  #     PROB MIN RUN T MAX RUN T   PATH DESCRIPTION
-------------------------------------------------------------------------

  1 0.00291     6654.0     6778.0  106->107->108->109->110->111->112->113
                                       ->115
  2 0.01000      231.0      231.0  106->114->115
  3 0.28524     5146.0     5266.0  106->108->109->110->111->112->113->115
  4 0.00679     6322.0     6412.0  106->107->108->110->111->112->113->115
  5 0.66556     4814.0     4900.0  106->108->110->111->112->113->115
  6 0.00003     6540.0     6660.0  106->107->108->109->110->111->113->115
  7 0.00003     6540.0     6660.0  106->107->108->109->110->111->115
  8 0.00294     5032.0     5148.0  106->108->109->110->111->113->115
  9 0.00294     5032.0     5148.0  106->108->109->110->111->115
 10 0.00007     6208.0     6294.0  106->107->108->110->111->113->115
 11 0.00007     6208.0     6294.0  106->107->108->110->111->115
 12 0.00686     4700.0     4782.0  106->108->110->111->113->115
 13 0.00686     4700.0     4782.0  106->108->110->111->115
 14 0.00003     6654.0     6778.0  106->107->108->109->110->111->112->115
 15 0.00288     5146.0     5266.0  106->108->109->110->111->112->115
 16 0.00007     6322.0     6412.0  106->107->108->110->111->112->115
 17 0.00672     4814.0     4900.0  106->108->110->111->112->115

-------------------------------------------------------------------------

PROCESSOR LOADING  =  2.168960E-01 SECS/SEC  OR  = 21.69 % OF CAPACITY


MEMORY OCCUPANCY
           PERMANENT  = 1994 16-BIT WORDS
           TEMPORARY  =   53 16-BIT WORDS
**********************************************************************
```

Figure 175. INSNAV Sizing Report (Sheet 2 of 2)

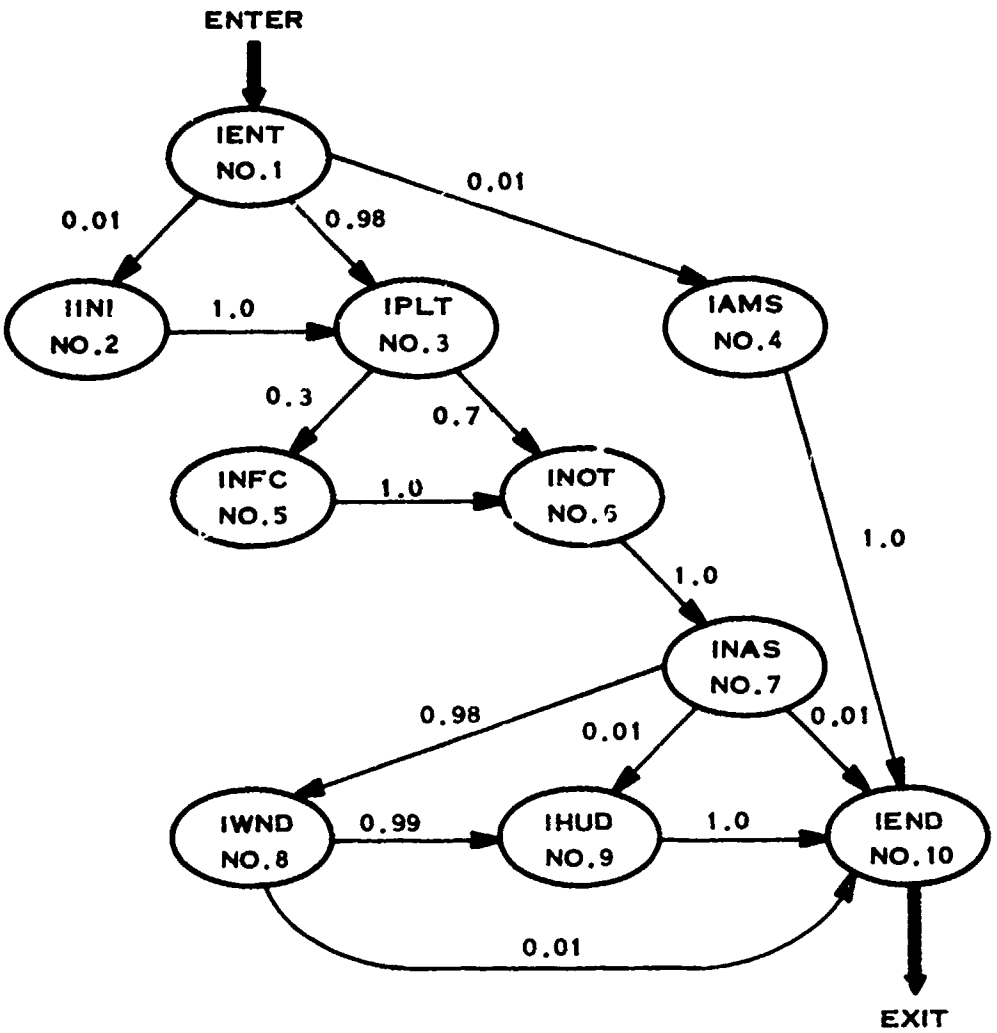



**Figure 176. Program Graph of INSNAV (Inertial Navigation)**

and resulted at most in a conservative approach to allocating application programs to DP/M resources. Table 36 represents this allocation. The avionics subfunctions which the application program names represent are shown in Table 37.

## 6. Simulation Description

This subsection describes the method and the procedure that were used to model the results of process construction for the study so the CAS mission could be simulated on the System Network Simulator described in Section VII. Model description input was made to the System Network Simulator consisting of data pertaining to: (a) bus characteristics; (b) Global bus connectivity specification; (c) Local bus connectivity specification; (d) task definitions; and (e) definition of all subfunctions scheduled by the Global Executive. The input format of this data has been described in Section VII.

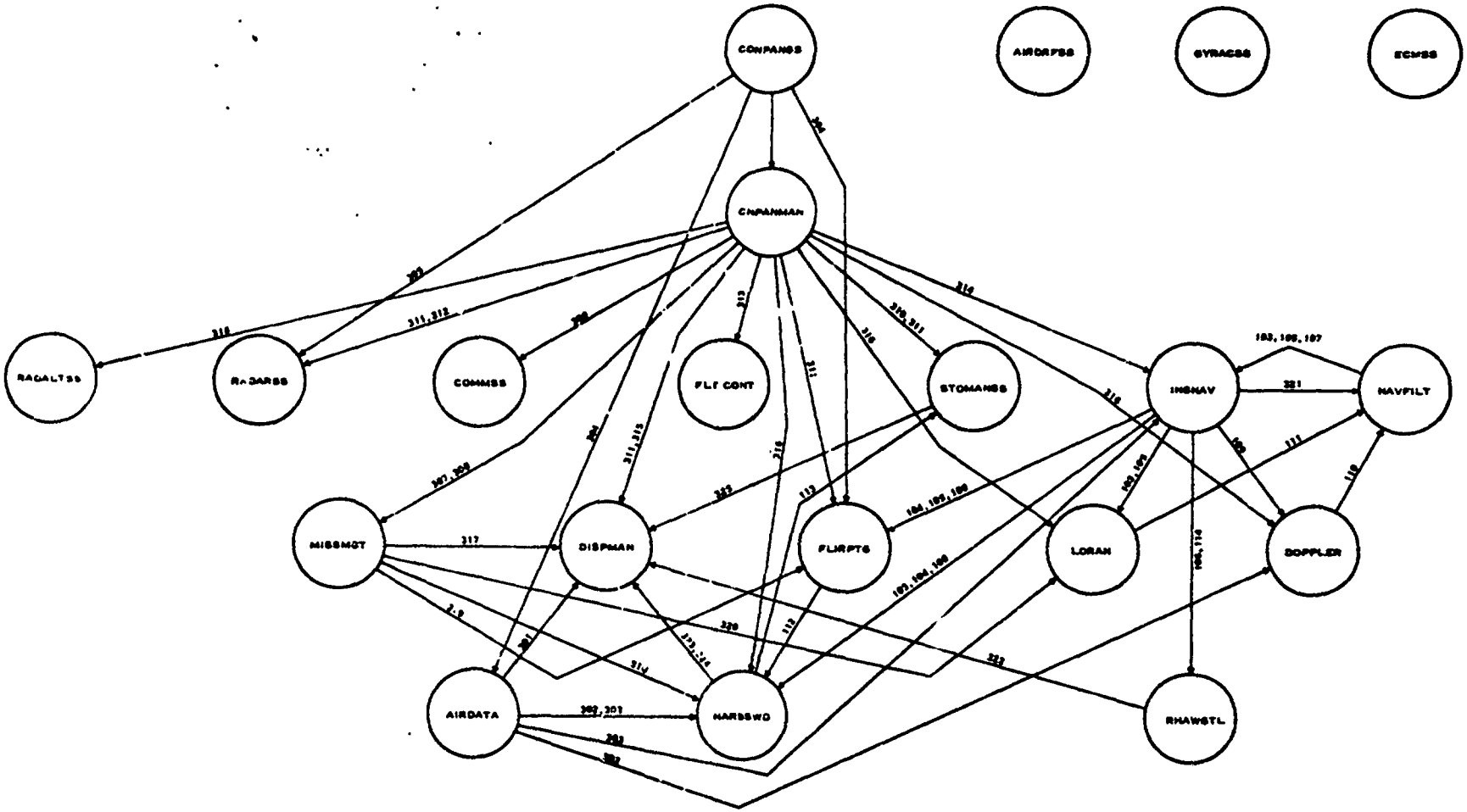
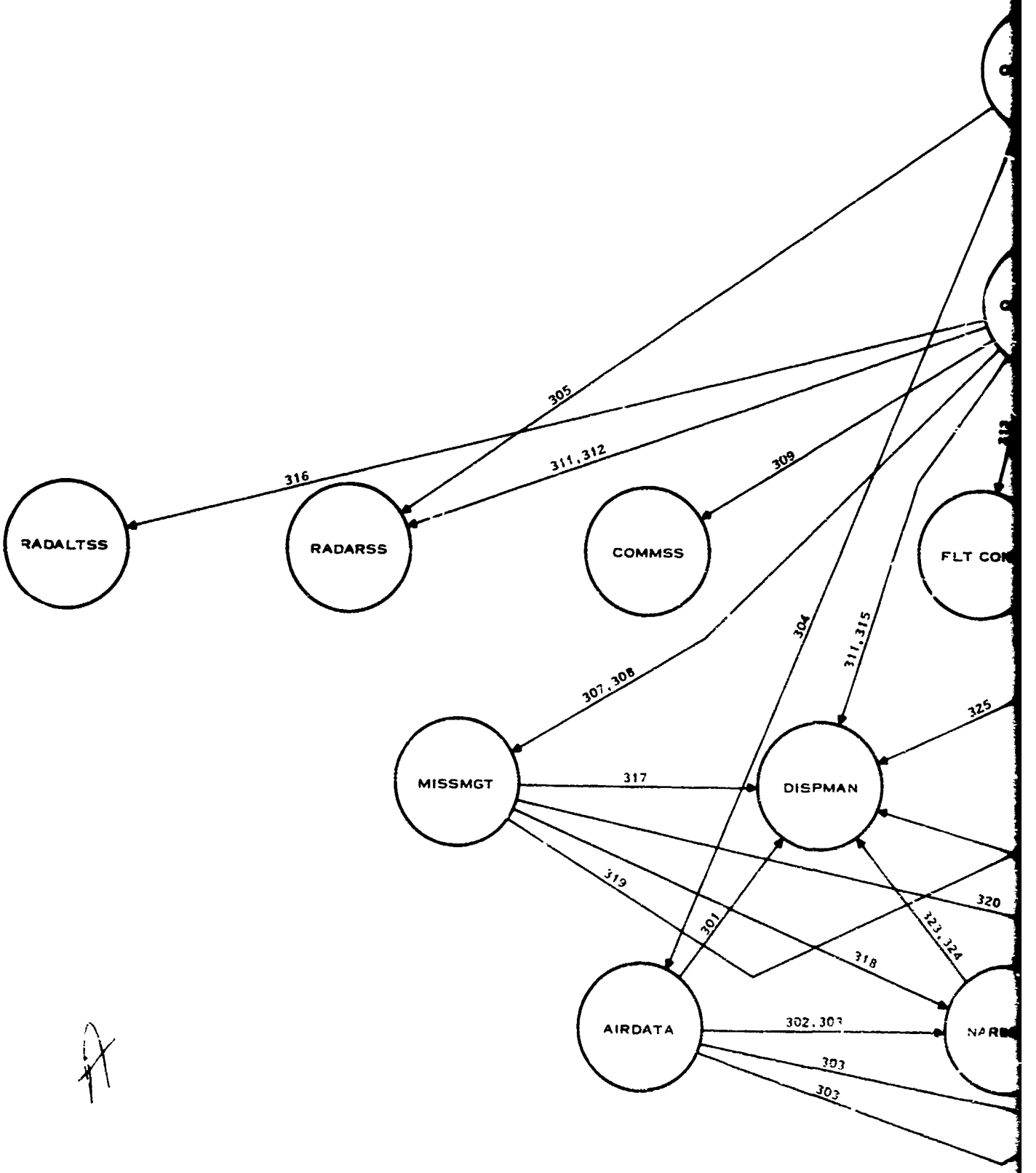



Figure 177 Data Study System Control and Message Flow


RADALTSS
RADARSS
COMMSS
FLT CON
MISSMGT
DISPMAN
AIRDATA
NAR
316
305
311,312
309
304
311,315
307,308
325
317
319
301
320
323,324
318
302,303
303
303

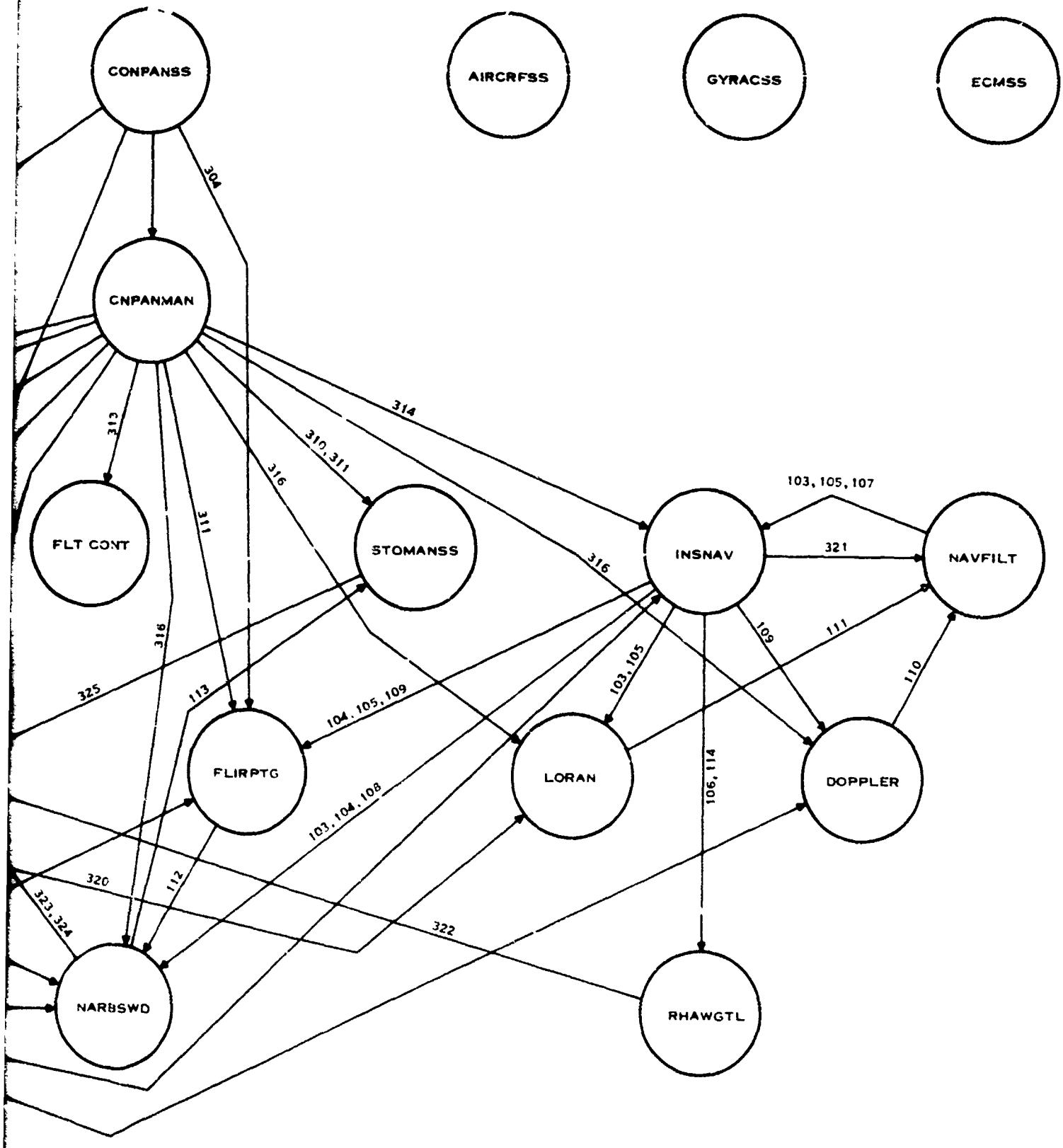



Figure 1[illegible] Case Study System Control and Message Flow

TABLE 36. PROGRAM ALLOCATION SUMMARY

| PE Number | Program Allocated |
|---|---|
| 1 | GEX |
| 2 | AIRDATA, MISSMGT, NAVFILT, AIRCRFSS |
| 3 | CNPANMAN, CNPANSS |
| 4 | DISPLMAN |
| 5 | FLTCONT |
| 6 | RADARSS |
| 7 | RADALTSS, GYRACSS, DOPPLER |
| 8 | ECMSS, COMMSS |
| 9 | STOMANSS, ENDSTO |
| 10 | STOMANSS |
| 11 | INSNAV |
| 12 | LORAN |
| 13 | RHAW |
| 14 | FLIRPTG |
| 15 | NARBSWD |

Table 38 provides the task IDs that were allocated to the various application programs that were used in this case study, the memory resource requirements as obtained from the simulation, and the execution time for each task, also obtained from the simulation. This information plus I/O data that was analyzed from the output of the process constructor was used as input data parameters to the task definition procedure for the SNS as described in Section VII.

Table 39 provides IDs of subfunctions, their iteration period, and their time of initiation relative to one another. These subfunctions are the programs which must be time-scheduled by the GEX. The subfunction definitions provide a time-ordered-list for the GEX for this purpose. Certain messages that are transferred over either bus in the DP/M system do not have successors. To elaborate, these messages do not contain the predecessor count of their destination task. They are required by the task, but are not critical in terms of timely arrival for the task execution. Then, messages generally are transmitted at rates slower than the iteration rate of their destination tasks. In the simulation, these messages are allocated IDs from 300 up.

### 7. Simulation Results

The reports generated of a simulation run for the case study are summarized in the reports found in Appendix B.

## E. RECOMMENDED FUTURE ACTIVITY IN DP/M PROCESS CONSTRUCTION

The objective of this subsection is to make several recommendations for future R&D work in DP/M process construction. These recommendations are based on the experience accumulated during the course of work described in this report. As mentioned in the discussion of automation of process construction (Subsection IX.B.3), the preparation of process construction inputs and the conversion of its outputs to simulation (or to process linking/loading) inputs were foreseen to be the potentially biggest bottlenecks in the entire process construction procedure. The experimental work carried out within the framework of process construction study has confirmed this suspicion. The means for eliminating these bottlenecks are within the present state of the art and are economically feasible. Hence, it is recommended that the early and late phases of the process construction procedure be automated by using the approach outlined below.

Preceding page blank

**TABLE 37. GLOSSARY OF APPLICATION PROGRAM NAMES**

| | |
|---|---|
| AIRDATA/AIRDATASS | AIRDATA, AIRDATA SUBSYSTEM SERVICE |
| CNPANSS | CONTROL PANEL SUBSYSTEM SERVICE |
| CNPANMAN | CONTROL PANEL MANAGEMENT |
| DISPLMAN | DISPLAY MANAGEMENT |
| DISPLSS | DISPLAY SUBSYSTEM SERVICE |
| FLTCONT/FLTCONSS | FLIGHT CONTROL/FLIGHT CONTROL SUBSYSTEM SERVICE |
| MISSMGT | MISSION MANAGEMENT |
| NAVFILT | NAVIGATION FILTER |
| RADARSS | RADAR SUBSYSTEM SERVICE |
| RADALTSS | RADAR ALTIMETER SUBSYSTEM |
| ECMSS | ECM SUBSYSTEM SERVICE |
| COMMSS | COMMUNICATION SUBSYSTEM SERVICE |
| AIRCRFSS | AIRCRAFT STATE SUBSYSTEM SERVICE |
| GYRACCSS | GYROS/ACCELEROMETER SUBSYSTEM SERVICE |
| STOMANSS | STORES MANAGEMENT SUBSYSTEM SERVICE |
| INSNAV/INSSS | INERTIAL NAVIGATION/INS SUBSYSTEM SERVICE |
| DOPPLER/DOPPSS | DOPPLER/DOPPLER SUBSYSTEM SERVICE |
| LORAN/LORANSS | LORAN/LORAN SUBSYSTEM SERVICE |
| RHAWGTL | RHAW SERVICE |
| FLIRPTG/FLIRSS | FLIR POINTING FLIR SUBSYSTEM SERVICE |
| NARBSWD | NARBS WEAPON DELIVERY |

### 1. Process Construction Inputs

The manual work needed to analyze the model data and subsequently to prepare the process construction inputs can be reduced by

- Introducing an extended language processor which, besides being a compiler in the ordinary sense, would have the capability to analyze the source program text, extract the needed model data, and produce the representations of this data in the form directly usable by the Process Constructor.
- Mechanizing the handling of model data through the facilities of a computerized data management system, discussed in Subsections IX.B and IX.C.

Several possible ways for representing the process model to the process constructor are discussed in Subsection IX.B.1. What has been proposed for the Basic Process Constructor constitutes a relatively simple but restricted approach. Although this method is sufficient for the procedure specified in Subsection IX.C, it is recommended that alternative and possibly more

TABLE 38. TASK ID, MEMORY AND EXECUTION TIME SUMMARY

| Task Name | Task ID | Memory Required | Execution Time (μs) |
|---|---|---|---|
| AIRDATA/AIRDATASS | 41 | 1034 | 4406 |
| CNPANSS | 43 | 747 | 11340 |
| CNPANMAN | 44 | 561 | 1387 |
| DISPLSS | 45 | 324 | 1630 |
| DISPLMAN | 46 | 2183 | 3670 |
| FLTCONT/FLTCONSS | 47 | 2129 | 5406 |
| MISSMGT | 49 | 41 | TBD |
| NAVFILT | 50 | 1715 | 228155 |
| RADARSS | 51 | 1240 | 2739 |
| RADALTSS | 52 | 391 | 883 |
| ECMSS | 53 | 480 | 1682 |
| COMMSS | 54 | 580 | 5072 |
| AIRCRFSS | 55 | 195 | 2628 |
| GYRACCSS | 56 | 114 | 738 |
| STOMANSS | 57, 58, 157 | 5928 | 8002 |
| INSNAV/INSSS | 59 | 2529 | 9121 |
| DOPPLER/DOPPSS | 61 | 715 | 2531 |
| LORAN/LORANSS | 63 | 2625 | 13656 |
| RHAWGTL | 65 | 766 | 23130 |
| FLIRTG/FLIRSS | 67 | 2249 | 7374 |
| NARBSWD | 69 | 3160 | 17469 |

general and powerful methods be investigated. The basic objective of these investigations would be to come up with model representation forms which

- Could be automatically generated as a by-product of compilation
- Would treat the process control dynamics and data flow by a single directed graph
- Would be relatively independent from the design alternatives for the system under development

**TABLE 39. SUBFUNCTION INFORMATION SUMMARY**

| Subfunction | ID | Iteration Period | Time of First "GO" Message |
|---|---|---|---|
| AIRDATA/AIRDATASS | 1 | 62,500 | 3,000 |
| CNP/NSS | 2 | 62,500 | 1,000 |
| DISPLSS | 3 | 250,000 | 2,000 |
| FLTCONT/FLTCONSS | 5 | 15,625 | 4,000 |
| MISSMGT | 6 | 1,000,000 | 9,000 |
| NAVFILT | 7 | 8,000,000 | 125,000 |
| RADARSS | 8 | 31,250 | 5,000 |
| RADALTSS | 9 | 31,250 | 6,000 |
| ECMSS | 10 | 31,250 | 8,000 |
| COMMSS | 11 | 62,500 | 15,000 |
| AIRCRFSS | 12 | 125,000 | 7,000 |
| GYRACCSS | 13 | 31,250 | 37,000 |
| STOMANSS | 14 | 62,500 | 0 |
| INSNAV/INSSS | 15 | 31,250 | 10,000 |
| DOPPLER/DOPPSS | 16 | 31,250 | 11,200 |
| LORAN/LORANSS | 17 | 31,250 | 43,650 |
| RHAWGTL | 18 | 31,250 | 11,600 |
| FLIRPTG/FLIRSS | 19 | 31,250 | 19,000 |
| NARBSWD | 20 | 31,250 | |

A survey of current literature [references (12), (14) and (15) are recommended as a point of departure for further research] suggests that Petri nets and related graphs may offer a medium for a combined representation of process control and data flow implied in the second requirement above. Especially, the so-called Macro E-Nets [reference (15)] appear to be capable of combining the control and data flow information under the same graph. Macro E-Nets are suitable for representing a wide class of parallel computational processes. Hence, they, in principle, meet the third requirement

[14] Miller, R.E.: "A Comparison of Some Theoretical Models of Parallel Computation," *IEEE Transactions on Computers*, Vol. C-22, #8 (August 1973).

[15] Noe, J.D., and G.J. Nutt: "Macro E-Nets for Representation of Parallel Systems," *IEEE Transactions on Computers*, Vol. C-22, #8 (August 1973).

Next, the question arises whether such graphs can be generated as a by-product of compilation. To our best knowledge, this still is an open problem, which should be further investigated. Two remarks are pertinent at this point:

Automatic generation of such graphs is not only a property of these graphs but also a characteristic of the language processors under consideration

Certain types of information, such as program scheduling rates, normally are not available at compilation time and hence must be independently provided by the process designer.

Thus, even in a highly automated process construction environment (as shown in Figure 178) there will be two input streams: one produced by the language processor; another, by the process designer.

## 2. Process Construction Outputs

Similarly, it is recommended that the final phases of process construction be more extensively mechanized and simplified than is stated in the design requirements for the Basic Process Constructor (Subsection IX.C). This can be done by defining a standard way to represent the models of a wide class of real-time processes for multiple computer networks. The present design of the DP/M hardware/software would constitute a particular case of such a generalized model class.

The advantages of such an approach are

Creation of a standard interface between the process constructor and the users of process construction outputs (various simulators, program linker-loaders)

Easier automation of the final steps of process construction.

A standard interface would serve a twofold function: first, it would make process construction relatively independent from the particular design of the Executive or from the software architecture of the system under development, secondly, process construction outputs would become available to the user programs in a more readily accessible form, which would also reduce the cost and time required to develop such post-process-construction support programs.

The right-hand side of Figure 178 shows the foregoing ideas on standardized outputs by showing a process constructor communicating with several post-process-construction user programs through a common Process Construction Output File. Typically, a consumer of the process construction outputs uses only a portion of the available data. For example, the memory maps used by the process loader are of no or little interest in simulation, on the other hand, much of the program execution timing data, which is of great interest in simulation, is not used in the execution of the process software on the target computer system. Hence, every user program must be provided with an input routine whose function would be to retrieve and reorganize appropriate portions of the Process Construction Output File.

Defining such a generalized, standard process construction output model for DP/M is a simple task. What is important is that it be defined before the design of process constructor and of other support software has progressed too far. In this respect, the situation is analogous to the development of system software: the composition and formats of object programs must be defined before the design of language processors and link-loaders can be completed.

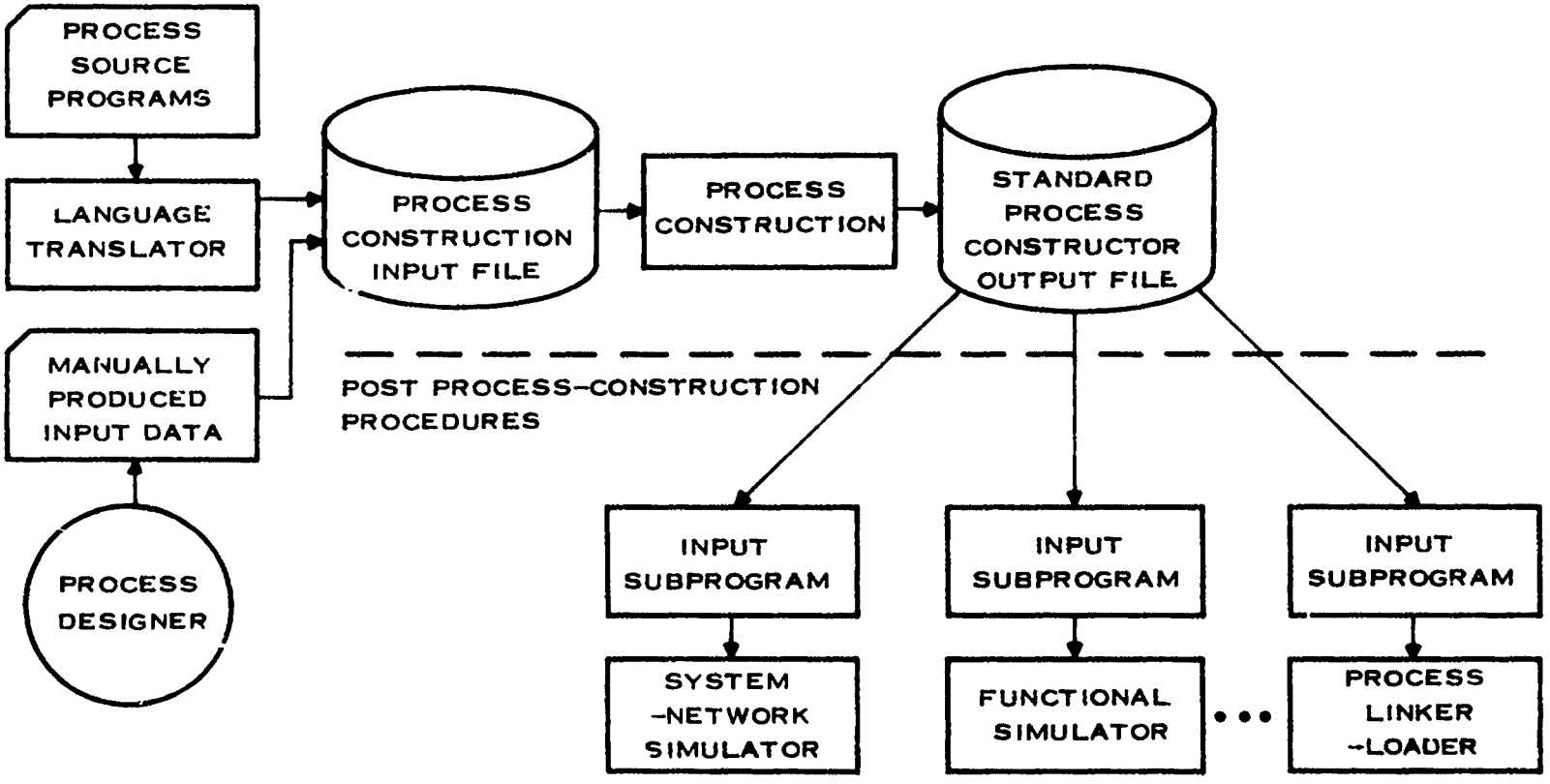



Figure 178. Automation of the Initial and Final Stages of Process Configuration

# SECTION X

# FAULT-TOLERANCE ANALYSIS

## A. INTRODUCTION

The goal of the fault-tolerance analysis task was to identify potential methods of exploiting the DP/M system architecture to provide overall avionic system fault-tolerant capabilities during the performance of an aircraft mission. It was recognized that the DP/M system philosophy is potentially amenable to techniques of graceful degradation to achieve functional fail-soft levels of fault tolerance, due to the absence of a complex central processing resource within the distributed-function system environment. Additionally, the low increment-of-capability concept used in DP/M system construction, with the use of low complexity and cost homogeneous microprocessor hardware, permits practical adoption of conventional hardware redundancy techniques. Thus, both functional and physical redundancies were identified as the two basic methods by which the DP/M design could facilitate overall system fault tolerance. Accordingly, the unique requirements of each method, with respect to the DP/M application, were investigated during system design. Each phase of functional element design was cognizant of this overall goal in an attempt to provide an accommodation of satisfactory fault-tolerance capabilities at each system component level.

A primary objective of the fault-tolerance-analysis activity was to initially identify and examine the various failure modes possible within the overall avionics system. More specifically, those failures which can be attributed to DP/M system components were identified and investigated in an effort to prevent the DP/M system itself from introducing any detrimental effects on overall system reliability, especially with respect to flight-critical system functions. As a complementary effort to the above objective of analyzing system failure modes and mechanisms, the fault-tolerance analysis also dealt with the task of identifying fault recovery techniques which could be effectively used to satisfy these requirements.

In summary, it was the intent of the DP/M fault tolerance analysis to identify, examine, and determine the potentially advantageous and disadvantageous characteristics inherent in the DP/M system architecture with respect to overall aircraft fault tolerance and reliability. Exploitation of the advantageous attributes and elimination of the detrimental attributes were underlying motives within this design intent. This analysis, accordingly, does not propose to develop an ultimate fault-tolerant system design which advances the state-of-the-art of fault-tolerant computing systems. Only the determination of those techniques which can be applied to enhance or use the inherent fault-tolerant capabilities existent in a generic DP/M system comprising a network of homogeneous PEs is sought.

## B. FAULT-TOLERANCE APPROACH

System reliability and fault tolerance are inherently interrelated attributes of a given system design, however, some confusion may exist concerning the effects of one attribute on the other. With respect to avionic system configuration, the term "reliability" is often used in different contexts. Simply stated, "reliable" has different meanings to different system users. At the user (pilot) level, reliability is used in conjunction with the probability of mission success. At the system component (maintenance officer) level, reliability is measured with respect to individual system component failure probability. Before the use of integrated avionic system

techniques the two interpretations of "reliable" were exactly equivalent, i.e., individual hardware component failures were directly reflected at the system user level by the attendant loss of some functional capability associated with the failed component. Integrated systems such as DP/M, however, can make a component failure transparent at the system user level via the use of redundant resources for automatic failure back-up and recovery. Hence, the term fault tolerance directly implies system resource redundancy when used in conjunction with DP/M-like systems. Fault tolerance does not entirely eliminate the need for reliable components; instead, it offers the option to allocate part of the reliability resources to the use of redundancy. Resource redundancy can generally be provided by two basic techniques: functional and physical redundancy implementations.

### 1. Functional Redundancy

Functional system-level redundancy within an avionics system is popular in present integrated avionics systems, as evidenced in the A-7D, F-111, F-15, and the proposed DAIS hot bench avionics systems. The integration of the various sensors, displays, and unique avionics devices offers the system architect multiple methods of performing different functions. As failures in a given subsystem are recognized, one of three alternative strategies may be used to "mask" the failure (completely or partially)

Derive failed function/equipment outputs required by other related system functions from an alternate source, thus prohibiting failure propagation to other dependent functions

Invoke an alternate mode of failed function performance at the probable expense of system performance (i.e., degraded mode)

Reallocate available system resources to the performance of all, or the highest, priority functions.

All failure recovery alternatives above indicate the implied association of potential overall system degraded-mode operation associated with functional redundancy techniques. These techniques fall under the category of "graceful degradation." Basic to the concept of graceful degradation is the idea that any particular system element normally performs certain fixed tasks, but can also assume additional, or different, tasks (inclusively or exclusively) when another element fails. An example of a simplified graceful degradation procedure applicable to an aircraft navigation subsystem is shown in Figure 179. In this example, the loss of a primary navigation subfunction due to associated sensor or PE failure causes the overall navigation processing to revert to an alternate, degraded mode of operation. All recovery actions involve only software mode switching in the operational PEs. Graceful degradation techniques attempt to mask system faults as much as possible, though not necessarily completely, from the system manager (pilot). In general, graceful degradation techniques offer the least costly approach to fault-tolerance capabilities. Addition of these techniques in system design imposes the requirements (1) that failures be detected, (2) that necessary memory and processing capabilities be available for back-up programs, and (3) that the reconfiguration software be developed, no hardware additions are required.

Another generic attempt at offering system-level functional fault tolerance is known as "dynamic resource allocation." This technique provides for switching of system elements, or resources, from task to task, to match changing work load and the availability of operational elements. Thus, "dynamic redundancy" is available for most processing functions. However, the hardware, and especially software, requirements of this approach are much more complex than

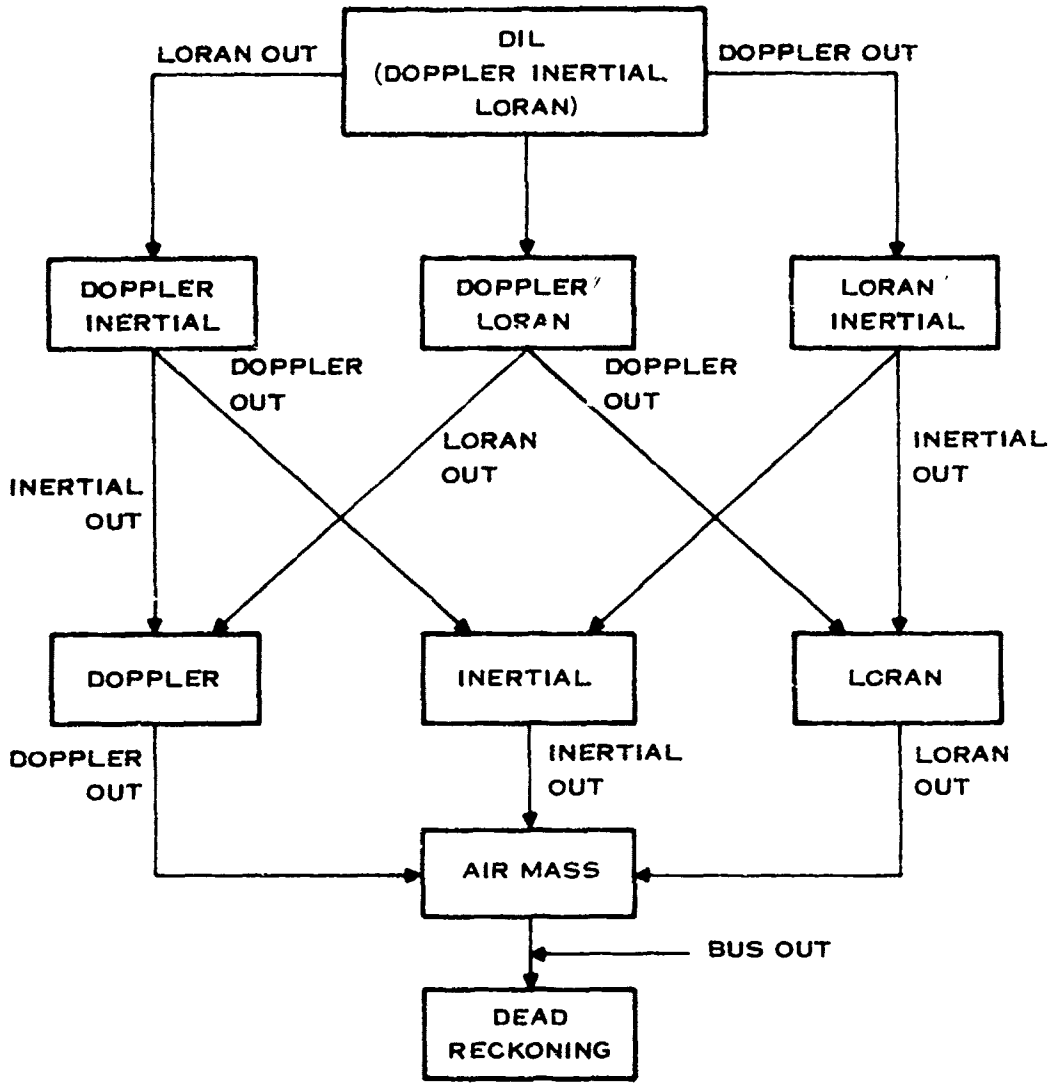



**Figure 179. Graceful Degradation of Navigation Function**

those associated with graceful degradation techniques, because the scheduling programs must continuously solve the problems of task and resource compatibility. Additionally, the resource allocation control processing degrades overall system performance capabilities available to applications task processing. These disadvantages forced the elimination of dynamic resource allocation system operation early in the system design.

## 2. Physical Redundancy

Physical redundancy techniques use replication of system hardware elements to achieve system fault tolerance. Redundant physical elements can be used as "dedicated" or "dynamic" functional back-up units. The use of dedicated function, or "simple" redundancy techniques is an obvious method of increasing mission success, but at the probable expense of increased failure rate and system costs, caused by added hardware complexity. Physical redundancy also requires

in-flight monitoring to determine which redundant element is operating incorrectly and to guarantee use of the healthy element only. Functionally, this monitoring can be achieved by either software- or hardware-implemented techniques.

Physical redundancy with hardware monitoring and redundancy management is readily justifiable for flight-critical functions because of safety-of-flight considerations. Such techniques are presently used in Fly-by-Wire (FBW) flight-control systems where process control elements are used with hardware "voting" mechanisms to achieve an ultra-high degree of fault tolerance. However, for functions which are non-flight-critical, the idea and application of hardware redundancy must be viewed carefully. Practically, the use of redundant hardware resources that have comparatively low system MTBFs may not be justified in terms of mission safety and system cost effectiveness. A component that is relatively low in terms of its system reliability usually has a number of parts that contribute to this unreliable nature. Consequently, the system can have a tendency to be self defeating: in an attempt to increase reliability via redundant hardware resources, the system may actually be less reliable.

Another point to consider with respect to simple physical redundancy involves the determining of which components of a function must be redundant. If one component has an MTBF that is several orders of magnitude higher than that of another component, it does not increase the overall system reliability to add hardware redundancy to the most reliable component without added redundancy to the less reliable component. Such is the case with airborne digital processors and most avionic sensor/actuator equipments. the mean-time-between-failures (MTBFs) of "sensors" range typically between 100 to 2,000 hours, whereas airborne digital processor MTBFs range from 1,000 to 30,000 hours for contemporary hardware with 10,000 to 100,000 hours anticipated for DP/M-like LSI PEs. Consequently, if the goal is to increase system or function reliability, it may be necessary to add redundancy to only part of the system.

## C. APPROACH TO FAULT SPOTTING

During the investigation of functional and physical redundancy techniques, it was apparent the key factor affecting the effectiveness of both redundancy techniques is fault spotting, or fault detection. In a system using non-self-repairing hardware components, faults must be detected and recognized by the system fault recovery "intelligence" (software) before recovery procedures can be invoked. Simple physical redundancy approaches are dependent upon the correct monitoring and detection of system faults when they occur, and proper determination or isolation of the faulty system element to allow selection of the correct, operational redundant unit.

Generally, system failures manifest themselves in one of two basic categories. "hard" or "soft." "Hard" failures include permanent equipment failures and are typified by physical hardware damage or defect. "Soft" failures is a term used to categorize transient or intermittent failures at any level of system operation. Representative of this class of failures are temporary data errors (e.g., bit error) caused by either the system environment (noise) or hardware faults caused by marginal component operation or design flaws. Software errors are also included in this category of failures.

"Hard" and "soft" failures must be discriminated by both the fault detection and failure recovery system processes to achieve true overall system fault tolerance. "Soft" failures are usually detected by data validity checks (hardware or software implemented) at various points in

the system. The system-level criticality and/or error susceptibility of the data being checked usually dictates the method of validity-checking used. Data validity checking techniques can range from software "reasonableness checks" to hardware use of self-correcting data codes. The detection of soft failures is necessary at the individual PE level to allow the various associated system software elements to locally "block" or "mask" erroneous data at the earliest possible level to prevent its potential propagation throughout the system. Detection of "hard" failures is necessary to allow defective component isolation so that proper redundancy management control (functional or physical) can be performed.

As a basic design precept, it was decided to ensure the fault-tolerant integrity of the DP/M system elements themselves, as a first step toward achieving overall avionic system fault-tolerance capability. Thus, the various potential failure sources within the DP/M elements were identified and, accordingly, the necessary detection mechanisms required for each fault were considered in the individual component designs. The resultant DP/M hardware design provides failure detection in every area of component/system operation that could be destructive to individual functions or system level operational integrity. All hardware-detected faults cause an associated interrupt to directly alert the PE software program of the fault condition. The purpose and operation of these fault-detection mechanisms have been delineated in previous hardware-design sections and need not be reiterated here. Instead, a summary of potential failures typical to the DP/M system and their associated detection mechanisms is presented in Table 40.

From an overall system viewpoint, the DP/M design supports continuous built-in-test (BIT) techniques of system operational integrity testing in both background and foreground processing modes. For example, each PE is provided with hardware mechanisms to support interruptible background Performance Assurance Test Software (PATS) during PE "idle" time, when no avionics processing is being performed. Also, special Bus Interface Unit hardware design supports simultaneous message transmission and reception capability (i.e., input/output rebound) to allow a software-controlled self-test of the BIU component normal operating characteristics.

Built-In-Test methods of fault detection and isolation may be extended beyond the individual PE level to the overall system level via the DP/M communications facilities. Global Executive (GEX) scheduling procedures accommodate the periodic scheduling of system-level PATS upon request and control of the ensuing "health-status" communications with each PE. Additionally, the BIU interrupt mechanisms allow immediate GEX response to locally detected errors/faults reported by individual PEs via the busing facilities. These locally detected errors/faults may be either PE failures or external I/O device failures. External aircraft device failures are detected via both the I/O interface hardware and software checks performed on I/O data (e.g., reasonableness). If individual I/O device characteristics permit, BIT communications with the I/O devices (i.e., with I/O device BIT) can be invoked by the system PEs, with device status interrogated at the local level and reported to the system error monitor (GEX) if failures are detected.

The preceding approach to detecting individual avionics equipment faults with the PE to which it interfaces may be advantageous with respect to overall system operational integrity. Due to the inherent preprocessing or "smart terminal" characteristics afforded by the DP/M system architecture, I/O device faults can be readily detected by the device-interfacing PE and thus prevented from possibly propagating throughout the total system; transient data errors can perhaps even be corrected (e.g., via re-issuance of the data transfer, use of software-handled error-correcting codes, etc.) by the PE before distribution to the remainder of the system, thus offering potential system fault tolerance to intermittent device errors/faults.

**TABLE 40 POSSIBLE DP/M SYSTEM FAILURES AND ASSOCIATED FAULT DETECTION**

| Failure Source/Type | Detection Mechanism/Technique |
|---|---|
| PE | |
| Memory Control | Memory Response Check (Timeout) |
| Memory Data | Parity Check |
| Illegal Memory Reference | Memory Response Check (Address Out-of-Range) and Write Protect (Optional) |
| Instruction Failure | Software BIT |
| Illegal Op-code | Firmware Detection |
| Arithmetic Overflow | Hardware Detection |
| Hang-up (Hardware or Software) | Missed Real-Time Schedule (Software Detected) |
| Incorrect Inter-PE Data | BIU (Software Detected) Data Validity Check Hardware |
| Bus Control Interface Operation | |
| Invalid Position<br>Bus Quiescence<br>"Chatty" Terminal | BIU Watch Dog Timers |
| Excessive Message Length | Global BIU Hardware Check<br>Local Software Protocol |
| Bus Data | |
| Environment-Induced Noise<br>Erroneous Supercommutation<br>Data BIT Error (Random, Burst) | Bi-Phase Encoding and Data Parity Checks |
| Added Bit, Dropped Bit | Message Bit Length Check |
| Message Identity Error | Bi-Phase Encoding<br>Parity/Missed Message (Software Detected) |
| I/O Device | |
| Sensor | Device Response Check, Reasonableness Test (Software), Device BITE Interrogation |
| Mass Memory | Validity Code Checks |

## D. FAILURE RECOVERY APPROACH

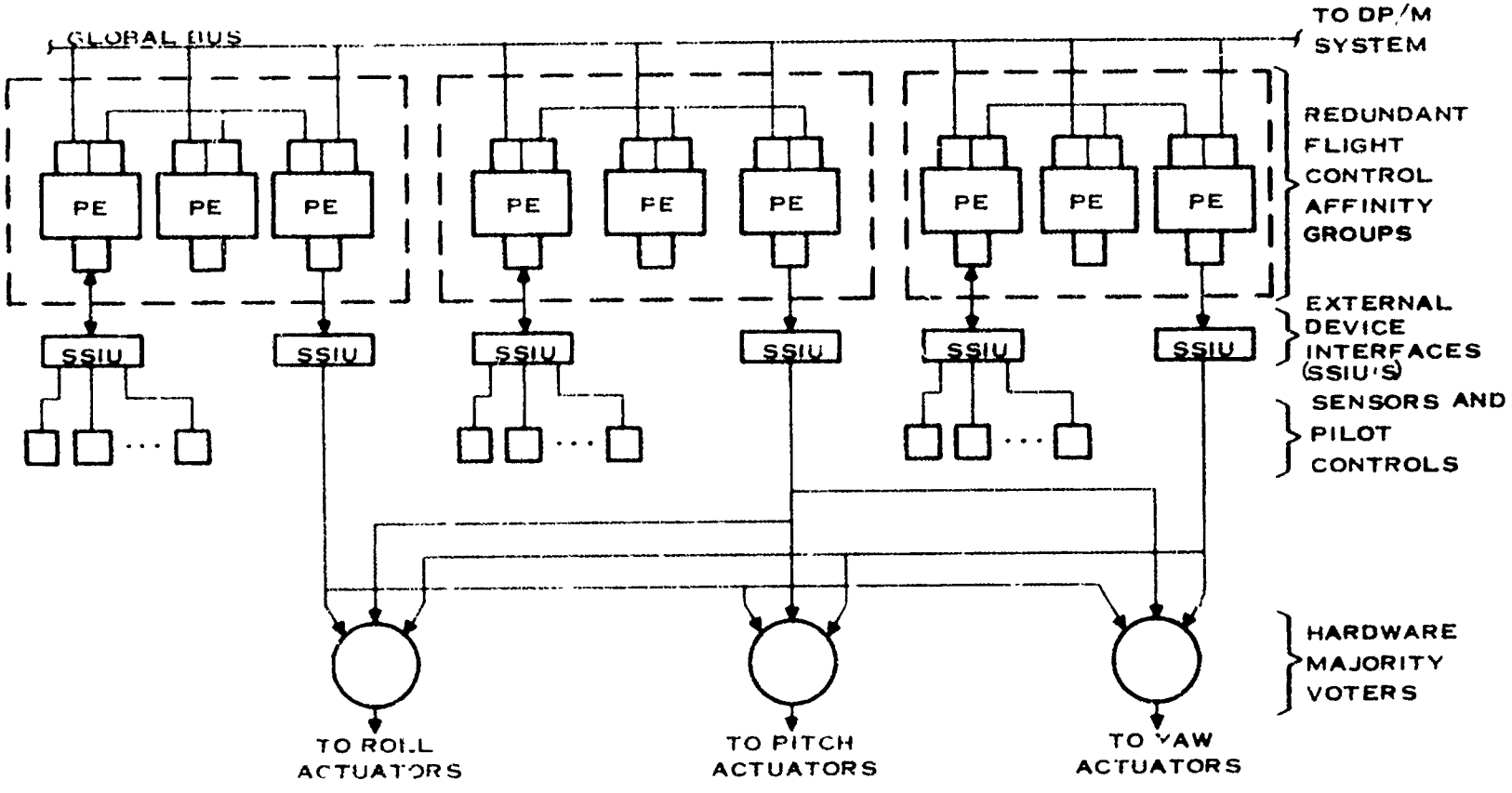
The DP/M system design appears equally amenable to the use of functional or physical redundancy techniques from the viewpoint that neither the system nor component designs contain any inherent attributes which functionally or physically preclude the use of either technique. The determination of which technique is more desirable is, however, dependent upon

the peculiar recovery requirements of each avionic function performed by the DP/M system during a particular aircraft mission, and thus a universal solution for every application is not realizable. However, certain guidelines or concepts can be identified.

The choice of recovery technique is extremely important to achieving satisfactory system tolerance to functional failures. The type of recovery technique permissible (desirable) for a given functional failure is directly dependent upon the redundancy method used to offer fault-tolerant capabilities to the performance of the function. The type of redundancy technique chosen is usually a trade-off decision between economics and function criticality, i.e., fail-soft operation (graceful degradation) versus fail-safe operation (absolute physical back-up, self-repairability). Once this decision is made, the primary factor in determining failure recovery technique in a real-time system is the required responsiveness of the system. This determination may, in turn, face reevaluation of the redundancy-type decision. To determine failure-recovery responsiveness of the avionic system to individual failures, the unique real-time requirements of each processing function must be analyzed to determine the amount of recovery time allowed before function performance is degraded to an intolerable level.

Generally, fail-soft recovery techniques are satisfactorily applicable to most mission-dependent functions; expensive fail-safe techniques are usually warranted only for functions where safety-of-flight considerations rule out any compromise in absolute favor of economics. Contemporary studies of avionics system designs have generally concluded that the only truly flight-critical function requiring absolute fail-safe operation in tactical aircraft applications is associated with FBW flight control. The currently recommended fail-safe approach to flight control system design is the use of triple or even quadruple redundant systems with failure detection and recovery accomplished concurrently with "hardware voting" mechanisms which allows complete masking of single (or double) system failures with no observable effect on real-time performance. It is recommended that a DP/M implementation of FBW functions follow this same approach, as exemplified in Figure 180. The DP/M PE design is amenable to this "hardware-voted" redundant system architecture. An Affinity Group required to perform all FBW processing functions would then be replicated the necessary number of times to achieve the desirable level of redundancy. The DP/M communications design facilitates this particular application since individual processing system isolation (functionally and physically) is achieved by Local bus and "dedicated" I/O device communications while communications which are required to be common between each individual system (and the remaining avionics system) are accomplished via the Global bus facility. It is also recommended that the PEs assigned to flight-control functions contain ROM implemented program memories to ensure operational integrity under circumstances of extreme power supply fluctuations or failure. This ROM implementation is not considered to introduce any disadvantage associated with system flexibility (as is the case with the conventional avionics suite) since flight-control algorithms (and, hence software programs) are fixed for a given airframe and are not likely to change after final development/performance validation over the life of the aircraft.

The remaining complement of aircraft mission-related functions appear amenable to fail-soft recovery techniques. Fail-soft recovery implies that the responsiveness of the system to individual errors or faults may cause transient or temporary degradation in system performance caused by execution of the recovery process. The particular failure recovery approach recommended for these functions requires the existence of a GEX software System Fault Monitor to initiate and control all system recovery actions. Basic to this approach is the detection of failures at both the local PE and GEX levels. Locally detected errors must be conveyed to the GEX via the Global bus. The GEX then analyzes the failure report and initiates




NOTE EXAMPLE CONFIGURATION SHOWN ARBITRARILY
ASSUMES 3 PE'S REQUIRED TO PERFORM ALL
FLIGHT-CONTROL PROCESSING

Figure 180 DP/M Dedicated FBW Flight Control Configuration (Triple-Redundant Example)

proper, predetermined recovery activity, if required. Normal GEX recovery actions would be to reduce real-time performance (e.g., the iteration rate) of some processing functions and possibly eliminate other functions altogether in graceful degradation recovery circumstances; or the GEX may deactivate a faulty system device and activate its physically redundant counterpart in managing simple physical redundancy.

Redundancy control can be implemented at the local level for external device failures where necessary equipment redundancy exists. Knowledge of total system status, however, is still maintained by the GEX via failure reports or active failure detection by the GEX itself for future recovery or post-flight analysis purposes. In its fault-tolerance role, the GEX is thus intended, primarily, as a system health-status monitor and system-recovery controller for DP/M system component failures. Component failures dealt with by the GEX are generally PE failures or busing facility failures.

It has been recognized that since the Global bus facility is the only "central" resource in the DP/M system and since its operational integrity is imperative for system fault-recovery activities, it should be fault-tolerant within itself to the fullest degree practicable. This resulted in the dual-redundant Global bus approach and functional design discussed in Section III. The system-level management and control of these redundant global buses is programmable via the GEX. A comprehensive complement of bus-fault detection mechanisms are provided in the BIU hardware to allow individual PE determination of bus-related failures. It is intended by design that bus-data errors be detected at the individual data recipients (PEs) with any error detection in a message transmission reported to the GEX. The GEX can then request message retransmission or take other action as appropriate. For major busing-facility-related faults which impair the operation of the entire facility (e.g., bus quiescence or dominance conditions), the GEX assumes both detection and recovery responsibilities. Detection is performed by special BIU hardware with the additional recovery aid of automatic halting (disabling) of the faulty busing facility operation so as to allow subsequent GEX-driven system recovery from a known, controllable state. This automatic bus "shut-down" is also favorable in the event of bus dominance ("chatty" terminal) failures since it offers a first-level attempt at ceasing bus-dominating activity in a faulty PE.

The bus failure recovery process primarily involves establishing individual PE (i.e., BIU) operational integrity via the alternate Global bus path until the faulty (fault-causing) PE is found. This procedure is effected by GEX control (via bus communication) of the BIU hardware elements provided in each PE to allow switching between the alternate Global bus path connections. For bus quiescence errors, this process may be eliminated or expedited since the "position" of the bus-controlling PE (i.e., the probable PE in error) may be readily extracted from "global bus state" information resident in the GEX's own BIU Global Bus Position Register (GBPR). Dominance errors, however, will typically require a "chit-chat" communication procedure with individual PEs to isolate the faulty PE. Once the faulty PE is determined, it may be functionally removed from the system by methods of disabling the PEs operation in the system (i.e., commanded to "shutdown"), or if this method is unsuccessful because of nonresponsiveness in the failed PE, the faulty PE may be "exiled" from the system by removing it (or the remainder of the system PEs) to the unused, redundant global bus.

After fault PE removal has been effected, either functional or physical redundancy control (depending on unique system design) is accomplished via Global bus communication from the GEX to the affected PEs. Finally, the system state and bus configuration must be modified and reinitialized to define the "new" global bus configuration before system restart (bus

synchronization) and recovery completion. Thus, much of the system failure recovery activity is similar to an abbreviated system initialization procedure.

The above recovery approach requires a variable and potentially significant amount of response time in execution. The actual response time incurred is a function of the type of failure and the size of the overall system; however, preliminary analysis and system simulation have shown that expected response times are relatively short (a few milliseconds) with respect to the execution period of most real-time avionics processing tasks (10 milliseconds to 1 second). Thus, performance degradation incurred because of failure recovery latency is expected to be extremely transient in duration, affecting perhaps a single iteration cycle in a real-time avionics process. Such errors tend to be "filtered-out" in a few subsequent processing cycles in most processes investigated, assuming restart from a known state is properly handled. This requirement to be in a known state is one justification for the GEX tightly coupled scheduling of subfunctions on a set of predecessor conditions that include "time to schedule." This "time to schedule" can also serve as a definite checkpoint of current system status and a possible departure point for initiation of system-recovery procedures. Additionally, the failure-recovery process may have invoked a degraded mode of operation for performing the failed function and/or other system functions, thus resulting in some system degradation anyhow. It is not normally believed that the relatively slow responsiveness of the system software recovery approach will significantly affect most mission-dependent functions. Those functions which cannot tolerate the processing latency incurred by the approach should be provided with hardware redundancy control and recovery mechanisms (e.g. flight control).

In summary, the DP/M design supports either or both physical and functional fault recovery techniques without basic system design modification. Also, "design-to-cost" and "pay-as-you-go" design philosophies are accommodated by supplying a minimal amount of fault-tolerance aiding mechanisms in the basic system component design. This approach allows the construction of various degrees of fault-tolerant systems at the option of individual system designs, paid for only by those systems which require the increased complexity.

## E. POWER SUPPLY CONSIDERATIONS

The DP/M processing system must derive its power from the types of primary power generated onboard the aircraft. Primary aircraft electric power generally is available as 120/208V, 400 Hz, three-phase alternating current and 28 VDC. Primary power is usually generated by one source with at least one backup source. For example, the A-7 derives its power from a master AC generator, a battery and a ram-air-turbine, while the F-4 has two AC generators. The characteristics of the various power sources available onboard the aircraft are defined and governed by MIL-STD-704A.

Several power buses are implemented onboard the aircraft to route the generated power to devices situated in different locations. For example, there are primary and secondary AC and DC buses, battery buses, and emergency buses distributed throughout the aircraft. A device is connected to one or more buses, depending on the criticality of its function. Such devices can obtain their power from multiple backup sources, if needed. Note that the assignment of power buses to individual avionic equipments is determined by the criticality requirements of functions using the equipments. These requirements are, in turn, aircraft- and mission-dependent.

The DP/M power supply(s) must provide satisfactorily regulated DC power, in accordance with individual PE input power requirements. It must also be tolerant of the power transient

characteristics defined in MIL-STD-704A. Since the DP/M system Processing Elements can be implemented with potentially volatile semiconductor memory devices, system fault tolerance to power-failure conditions dictates effecting nonvolatile memory operations at the system level. This nonvolatility characteristic can be achieved by either use of nonvolatile mass memory for a system memory reload capability, or actually achieving nonvolatility at the individual memory level by providing an "ultra-reliable" power source for the memory elements. The latter approach is recommended for DP/M as discussed below.

The "ultra-reliable" power supply must be capable of supplying continuous, uninterrupted power to the PE within PE tolerance limits. Since the primary power source may experience intermittent periods of complete power loss (e.g., during generator failure), an "ultra-reliable" power source must possess a capability of energy storage or generation to provide continuous power output during these relatively short periods of primary power loss. This required capability implies the use of a backup battery in the PE power supply unit(s). Battery-supplemented power supplies are not new to digital processing equipment design, as evidenced by the current trend to use such techniques in many contemporary commercial minicomputer systems using semiconductor read/write memories. The use of batteries in such systems is usually accompanied by necessary sacrifices in system size, weight, and complexity (cost). These disadvantages are usually of little or no consequence in the commercial systems applications mentioned, but may present a situation of significant concern when applied to the tactical aircraft environment. However, the low-power consumption properties of an LSI all-semiconductor PE facilitates the use of "ultra-reliable" power supply techniques and greatly alleviates the above concerns associated with battery-supplemented power supplies in the aircraft environment.

An investigation of the aircraft power environment specified in MIL-STD-704A indicates that the energy-supplying capabilities of the backup battery device must be sufficient to sustain PE-required power levels during periods when primary aircraft power-supply voltages are either disrupted completely (i.e., complete power failure) or temporarily exceed normal steady-state limits (e.g., power out-of-tolerance transients). The primary aircraft power transients condition specified in MIL-STD-704A for DP/M class equipment are shown in Figures 181 and 182 for AC and DC power use, respectively. The power transient characteristics shown indicate a possible worst case abnormal (out-of-tolerance) voltage condition duration of approximately 7 seconds. Therefore, the PE power supply battery element must, as a minimum, be capable of sustaining PE power input during this "possible" transient time. However, further consideration of maximum possible abnormal power level time duration requires the battery backup to supply PE power for an approximate 30-second period in extreme "abnormal" circumstances; this more extreme requirement is caused by maximum anticipated time delays required to switch from primary power to backup power in the event of total primary/secondary power generation failure (e.g., time to physically deploy ram-air-turbine backup system) and also to switch from ground-power to aircraft power during flight-line power-up initialization and checkout activities, without requiring reinitialization of PE memories.

The above environmental constraints placed on the "ultra-reliable" power supply operation may be used in conjunction with anticipated PE power requirements to arrive at a quantitative estimate of actual battery requirements and, hence, the resultant impact on power supply implementation complexity/practicality. Determination of PE power requirements during system (aircraft) power failure conditions indicates that as a minimum absolute requirement, backup power is necessary only to permit retention of memory data. The remaining functional elements of the PE do not necessarily require continued power supply during periods of aircraft system

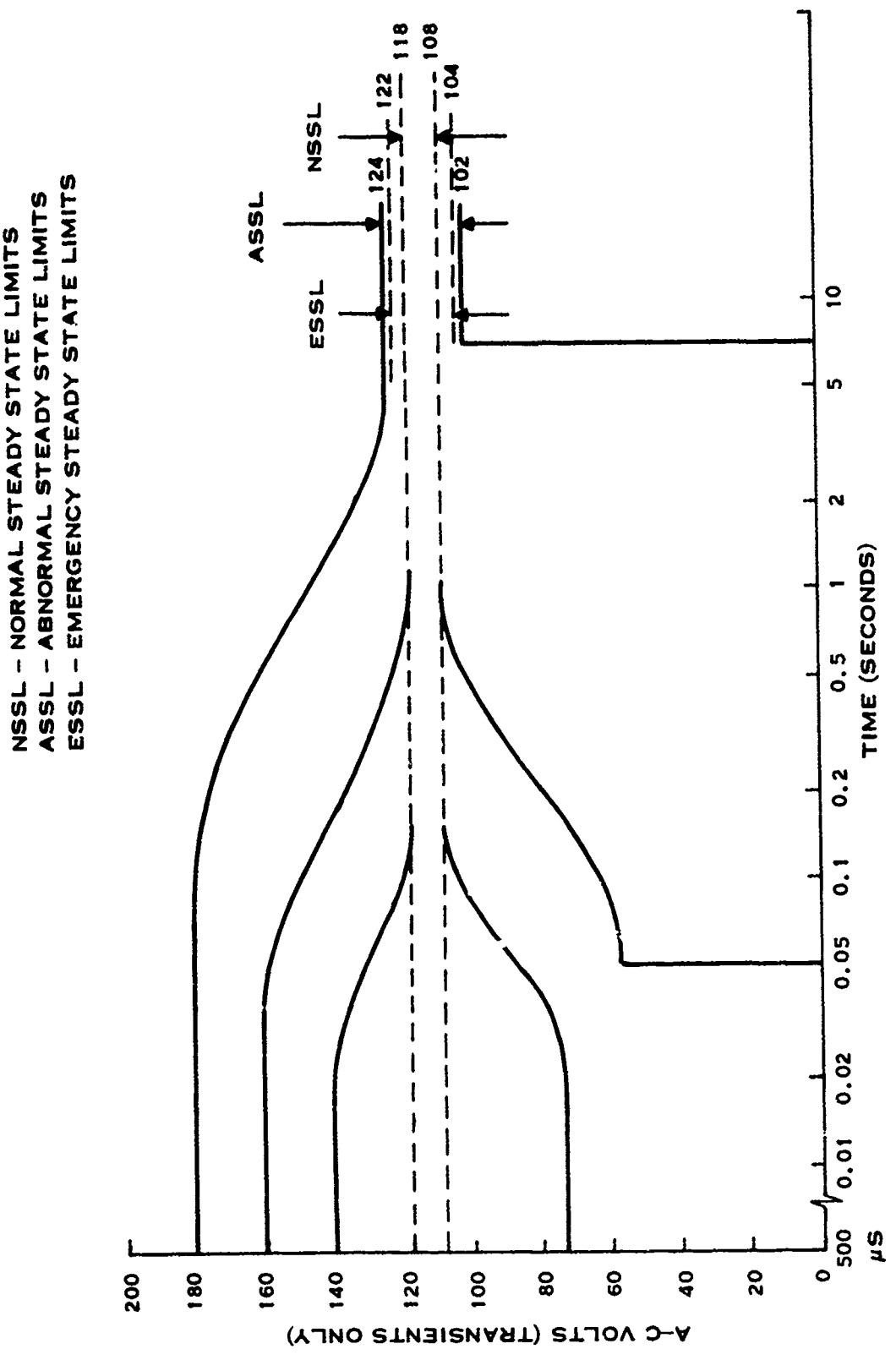



Figure 181. Transient Surge AC Voltage Step Function Loci Limits For MIL-STD-704A Category B Equipment

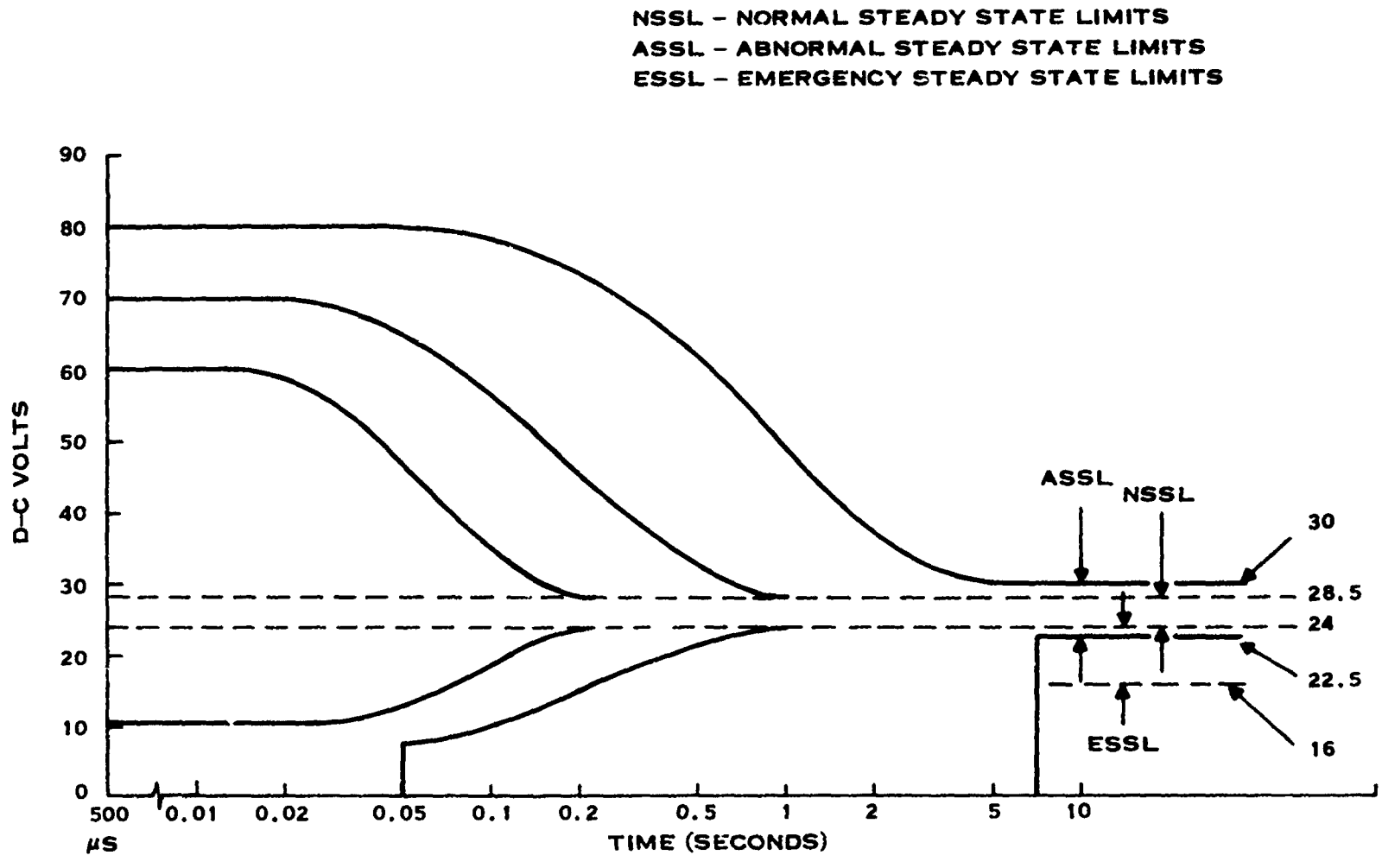



Figure 182. Transient Surge DC Voltage Step Function Loci Limits For MIL-STD-704A Category B Equipment

power failure, since, from each individual PE viewpoint, PE functional utility is eliminated when associated aircraft equipment is inoperative.

However, system-level considerations may indicate the desirability to retain PE operability during "system-down" time. One such consideration is the minimization of system recovery time following power resumption. A complete system initialization in response to each system power failure (transient or permanent) is obviously undesirable because of the requirement for an "onboard" program-load device (nonvolatile mass memory) and relatively long system initialization times incurred. This complete system initialization requirement is eliminated by the recommended concept of "ultra-reliable" PE power supplies which prevent PE program destruction and obviate any program reload requirement. However, partial system initialization would still be required to reinitialize or reestablish volatile system control parameters (i.e., register contents) such as those required for inter-PE bus configuration control and processor unit program registers. Further system level considerations reveal the potential impact of the loss of power in a single PE or Affinity Group in the operation of the overall system communications facilities. Since each PE connected to an information transfer bus must actively participate in the distributed bus control procedure, the failure of a "powered down" PE to participate could suspend remaining system operations until an abbreviated system-level bus "reconfiguration" is performed to remove the inoperative PE. In addition, when power is restored, the affected subsystem must be "brought back" into the system by another "bus configuration" system control procedure, again temporarily suspending other functionally independent processes.

Depending on individual function criticalities with respect to tolerating possible suspensions in operation and the potential or expected frequency of aircraft power transient faults, the bus reconfiguration activities mentioned above may be undesirable. In an esthetic sense, at least, it does not appear desirable to allow a failure in one system function to affect the operation of unrelated functions elsewhere in the system.

A possible solution to this problem is the mandatory attachment of each DP/M power supply unit to the same levels of aircraft primary power buses. This would, however, dictate that each PE be supplied power from the source(s) required for the most critical PE function; the requirement may be unreasonable or impossible because of the usual relatively limited power output capability of the tactical aircraft backup and emergency power sources. An additional disadvantage is the constraint it places on the topological distribution of the various power buses throughout the aircraft. This constraining characteristic would violate the DP/M system design goal of permitting and supporting physical distributiveness.

An alternate solution is the existence of an "ultra-reliable" power supply for every system PE (regardless of functional assignment or memory implementation) to support the bus control activity during power transients. This would eliminate the possible situation of individual PE or Affinity Group transient power failures. The only power failures affecting a portion of the overall system would be permanent failure of either the associate DP/M power supply or the primary aircraft power source to which it is attached.

Investigation into the actual implementation of this solution was performed with the primary goal of minimizing the battery backup power supply requirements. No justification for retaining Processor Unit and Input/Output Interface Unit operation is necessary. The only functions remaining fully operational are those associated with the bus control activity in the PE's Bus Interface Unit. The memory, of course, requires enough power to sustain data retention as mentioned previously. In particular, the operational status of the BIU need be preserved only

for the Global bus interface. Local bus activity is associated strictly with intra-functional activity and is thus not a factor in the system-level problem being addressed.

Thus, during power-fail conditions, only the PE memory unit and Global-bus-control-related portions of the BIU require continuous backup power; the remainder of the PE hardware components are allowed to "power-down." During the absences of satisfactory primary power and upon return of satisfactory power levels, the system PEs are forced into a CLEAR/RESET state by the power-detection logic within the PE power supply unit. The initialization control section of each PE then controls immediate PE operation following the removal of its CLEAR/RESET signal just as it does during a complete system initialization procedure (initial system turn-on). The initialization control (bootstrap) program is capable of distinguishing a system restart condition from a system-initialization condition and directs PE activity accordingly (i.e., all system program load operations in the normal system initialization procedure are bypassed in the power resumption/restart procedure.

Estimates of the energy storage required to power the PE memory over the 30-second maximum aircraft power failure require quantitative knowledge of unique device power consumption characteristics. The candidate semiconductor read/write memory technologies recommended for DP/M application are all inherently low power in normal operation, with extremely low-power dissipation characteristics attainable by "standby" modes of operation. Worst case standby mode estimates for the various candidate technologies reveal estimated power consumption levels on the order of 50 microwatts per memory bit. For an 8K memory configuration, approximately 0.46 watt of power will be required to support standby operation. This power consumption requirement converts to a battery supply capability requirement of 0.32 mA·hr, an extremely minimal requirement.

A quantitative estimate of BIU "standby" power requirements is slightly more complex, since actual power consumption may depend upon the physical partitioning of the BIU. Consequently, it appears desirable, for the sake of minimizing "standby" power consumption, to physically partition the BIU into multiple devices using either "normal" or "standby" power sources. Isolated power inputs may be provided to the device which implements the Global-bus-related control functions and other BIU functions to minimize superfluous power dissipation. As a worst case estimate, the required BIU standby power was assumed equal to the combined maximum power consumption of all basic Global bus-interface-related functional hardware (e.g., using the BIU device partitioning described in Section VI, the associated devices are 1-BILU, 2-BITU, 1-RBMU, and two line driver/receiver pairs). The total estimated worst case power requirement for these combined hardware functions is 1.26 watts. This power consumption converts to a battery energy supply requirement of approximately 12 mA·hr.

From the above estimated "standby" power requirements of the PE Memory and Global Bus Interface functional units, it appears that a battery with a minimum charge capacity of approximately 12.3 mA·hr will provide sufficient backup capability. This charge capacity requirement is extremely minimal and offers a preliminary indication of the practicality (physical and economic) of the approach. (As an example of representative battery requirements, present commercially available nickel-cadmium AA cells are typically capable of 400 to 500 mA·hr at 1.25 volts.) Actual battery requirements cannot be determined without detailed power supply design and various military environment battery characteristics being researched and analyzed.

Thus, it appears that the "ultra-reliable" power supply approach to achieving PE memory fault tolerance to power failures is amenable to low-cost miniature-size battery solutions.

introducing little impact of overall system hardware requirements. The only realizable impact of the above standby operation approach on DP/M pertains to ensuring operational integrity of the standby-powered hardware functions while other, interrelated hardware functions are inoperative. It is proposed, however, that standby mode management can be easily facilitated with proper use of the CLEAR/RESET signal generated from within the "ultra-reliable" power supply unit to the PE during the period of primary power failure. This signal can be used to denote standby mode operation, thus allowing the memory controller and bus access controller logic elements to perform operations related to standby mode activities only. During the presence of this signal, all normal operating mode signals/procedures are ignored (e.g., the Bus Interface Unit performs only the duties of "passing" bus control when its access slot is recognized). Thus, the functional design of the bus access control related hardware elements do not use the CLEAR/RESET signal as an asynchronous "reset." It should also be noted that from an implementation viewpoint this approach requires that the bus-access-control-related sequential logic functions be implemented with asynchronous techniques to prevent the otherwise attendant requirement for PE clock operation also during standby. Additionally, all interrlated, inoperative function signal interfaces of the standby-powered elements should be designed so that the logical value of such signals received are "FALSE" when unpowered. These hardware design impacts are not considered of great significance to hardware complexity; however, they will require a further amount of design consideration/verification to ensure correct device operation. As a departing note, the whole consideration of PE (specifically, BIU) "power segregation" may be strictly academic once an actual battery choice is determined. The entire PE could be operated in a standby mode (forced "idle" caused by the CLEAR/RESET signal) with an estimated battery charge capacity of approximately 50 mA·hr. Thus, battery-size availability may result in the use of a more than adequate capability, allowing simplified PE component design; or the incurred penalties associated with battery size, cost, and reliability due to the greater capacity requirement may be negligible.

The techniques of battery backup power supply implementation and usage are conventionally well known and should present little associated design risk. The functional attributes of the power supply design are represented in Figure 183. Furthermore, with proper physical implementation, ultra-reliable PE power supplies may be modularly configured from existing equipment ("standardized") power supplies, aircraft device power supplies may be easily upgraded to "ultra-reliable" configurations (for "smart sensors'), or the backup battery equipment may be eliminated from deployed power supply units as nonvolatile semiconductor memories become available and are used in DP/M systems.

Two schemes of potential physical deployment of the DP/M PEs and associated power supply allocation are currently envisioned. One scheme is to physically collocate a PE or an Affinity Group with a device which provides primary I/O to the function in that PE or Affinity Group. In this "smart-sensor" configuration, the PE/Affinity Group would derive its power from the same power source which supplies the device as shown in Figure 184. Therefore, both the device and Affinity Group would be affected in the same way in the event of a power failure. The capability would exist in this scheme to switch the device on and off, but the Affinity Group may require continuous power (i.e., ultra-reliable power), depending upon associated PE memory technology. The disadvantage is that, should an Affinity Group member PE be connected to another device which also is a primary I/O contributor to the function in this Affinity Group, but is not collocated, separate power supplies would have to be provided for each PE.

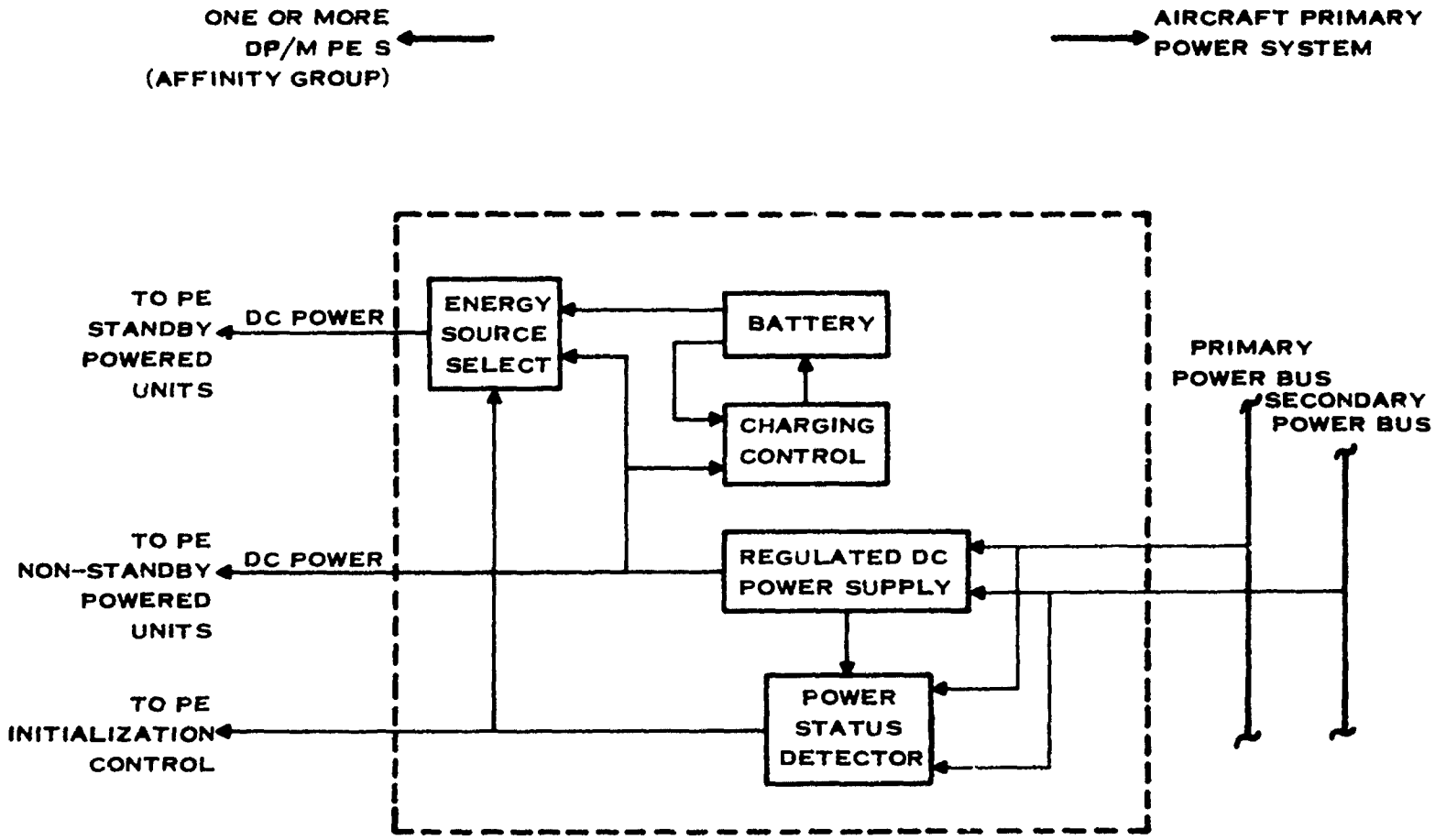



Figure 183. "Ultra-Reliable" PE Power Supply Functional Block Diagram

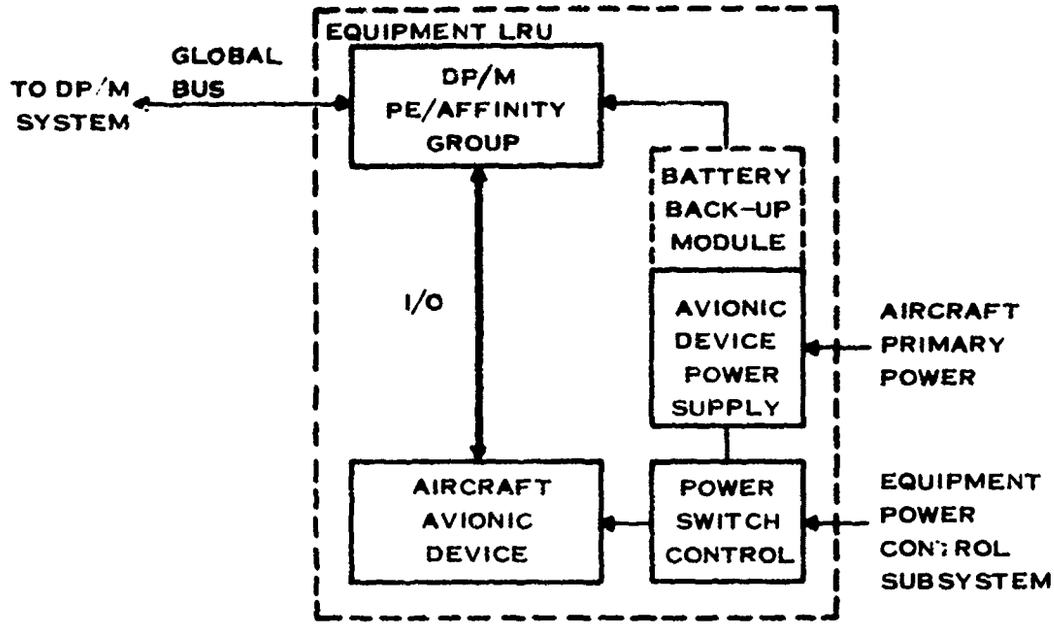



**Figure 184. Co-Located PE(s) and Associated Aircraft Device With Common Power Supply**

In the other configuration, Affinity Groups are suitably located in various parts of the aircraft so that they are close to the I/O device they service and yet can be connected to all the power buses so they can be supplied primary aircraft power, even in the most degraded backup mode. The disadvantage of this scheme is that some devices can be so located that relatively long I/O cabling may be required from PE to device. The obvious advantage is that the DP/M system is assured access to the most reliable elementary power system. Also, a power supply can be provided for each Affinity Group or for multiple Affinity Groups, thus improving space and cost economy. Power supply allocation to PEs (i.e., Affinity Groups) is dependent upon function criticality which dictates primary/secondary/emergency aircraft power bus access. Allocation option examples are shown in Figure 185.

On an aircraft with multiple power buses, there is one consideration in connecting a DP/M PE and its external "SSIU" device to a given sensor/actuator. If the SSIU must convert synchros or any ac transducers that require a reference voltage for conversion, the SSIU and sensor must be connected to the same power bus since one phase of the ac power is used as the common conversion reference voltage. This particular requirement may dictate the actual power connection configuration between a PE and the sensor to which it is interconnected. Similar limitations are likely to exist when a set of sensors is to be configured for a given suite of avionics.

In summary, the important observation with respect to DP/M power reliability is that it meet MIL-STD-704A, regardless of which power bus it is connected to. Conversations with tactical pilots and examination of current advanced aircraft system studies imply that the probability of primary power failure during a mission is almost negligible, and if primary power

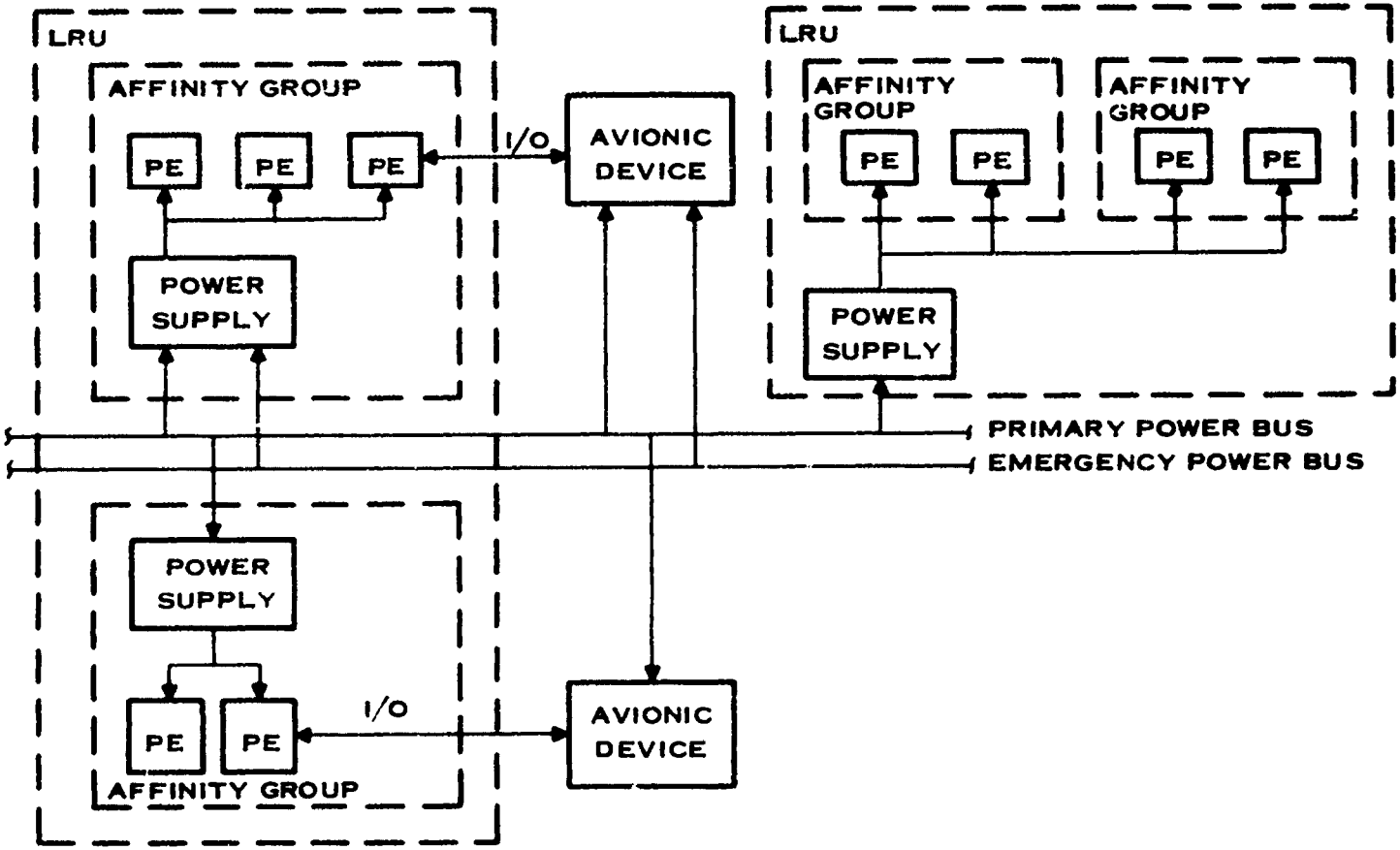



Figure 185. Physical PE/Power Supply Allocation Alternatives

does completely fail, the safety of the aircraft becomes of prime importance and the mission would most likely be aborted. An additional consideration exists if the aircraft uses a digital fly-by-wire system: processors, voters, and actuators must be supplied with a minimum of 30 minutes of backup power to permit return to base and the controlled landing of the aircraft. This requirement for sustained operation from a fixed backup battery source (aircraft primary battery power) encourages the use of low-power-consumption processing elements such as the DP/M.

## F. SUMMARY AND CONCLUSIONS

This paragraph summarizes the fault-tolerance analysis and assesses the impact of fault-tolerant design on DP/M system characteristics. The DP/M system that has evolved as part of this study incorporates within its design the concepts of fault tolerance and, as such, attempts to implement the realization of a total system fault-tolerant computing capability. The DP/M system as designed offers a number of significant advantages.

The system can detect faults and initiate remedial action before damage or damaging information propagates through the system, and thus offers the potential of truly graceful degradation. The low-cost homogeneous building-block concept of the system design allows redundant units to be added to the system to provide fault tolerance for critical computational functions.

A type of reconfiguration presently in use in multiprocessor systems and federated systems involves the reassignment and reallocation of hardware resources to functional responsibilities in response to various system component failures. This reconfiguration philosophy can impose special hardware requirements (e.g., shared resources, such as memory) and requires the development of complicated reconfiguration control software. Because of the dedicated nature of some of the DP/M system components, and the low projected cost of future LSI microprocessors such as the DP/M PE, reconfiguration as described above is no longer practical and is not supported by the DP/M system design.

As described in previous paragraphs, each aspect of system design has paid careful attention to the consideration of fault tolerance, so that the achievement of an effective level of system fault tolerance is integral to the system building-block components and will not require costly future "afterthought" add-on realization. The design of the functional hardware elements, the table-driven executive software structure, and the tightly-coupled global control concept with loosely-coupled local control are all design schemes that have evolved from the study of alternative tradeoffs to system-level fault-tolerance capabilities. The approach taken in the DP/M design has been to consider the problem of reliability and expandability from the initial stages of the system design.

Since the basic distributed functional microprocessor-implemented system architecture was determined to be advantageous to improving present system reliability and readily amenable to conventional fault-tolerance techniques, no real impact on system-level design was expected or encountered. The only impact experienced was increased hardware complexity in the PE design, caused by the necessary inclusion of certain fault-detection mechanisms to allow system-level fault tolerance via hardware fault detection and subsequent software-controlled recovery. However, detailed investigation into the amount of added complexity required for the detection mechanisms showed relatively little contribution to overall unit complexity with one possible

exception in the Bus Interface Unit. It was determined that elimination of the self-test feature of full-duplex BIU operation with the individual information transfer buses could result in a potential 25-percent reduction in the most complex BIU device type. It is, however, realized that this tradeoff cannot be decided until actual device technology capability versus the estimated requirements are determined, a function of the future time of device implementation.

In conclusion, the DP/M system offers basic architectural and component design features which are amenable to conventional fault-tolerance techniques. Variations in this technique are facilitated with basic fault-detection capabilities built into the DP/M component designs; with low complexity impact. Unique technique hardware requirements, however, must be added as dictated by definitized individual system fault-tolerance requirements. Thus, upgradable, "pay-as-you-go" fault-tolerant system configurations are allowed at the expense of only those applications where added cost is justifiable; the potential application of DP/M systems or components in fault-intolerant applications is not jeopardized by penalties incurred in defraying the costs required of the low-volume sophisticated fault-tolerant applications. Exact approaches to fault-tolerant system design characteristics, such as redundancy technique and fault recovery technique, are dependent on individual system requirements; however, it appears that the basic DP/M concept and design provides the necessary functional tools with which a wide variety and varying degrees of system-level fault tolerance can be effectively produced.

## APPENDIX A

## BASIC PROCESS CONSTRUCTION INPUT/OUTPUT DATA FILES

In the description of Process Construction input/output data files, the following terminology is used:

(1) The term "software module" denotes any software component under consideration (and also the corresponding segment of the execution-time code), which may be a whole program, a task (procedure) of that program, or a subprogram called by a task

(2) The term "data set" refers to the nonexecutable data of the process (such as individual variables, arrays consisting of such variables, records, or files) which requires storage at execution-time.

Pages 462, 463 and 464 are Blank.

FILE NAME: Computer-Independent Model of the Process (Input File)
DESCRIPTOR. Exogenous input to Step 1 of the Process Construction Procedure
FORM: Card images constituting a disk file
CONTENTS:

1. Process model ID/descriptor
2. Number of programs in the process model
3. For each program in the process model:
   - 3.1 Program ID/name and type code (External I/O Handler, avionics, etc.)
   - 3.2 Name of the avionics (sub) function which the program represents
   - 3.3 Scheduling parameters:
     - 3.3.1 Iteration rate
     - 3.3.2 Priority classification
     - 3.3.3 Other parameters not yet identified
   - 3.4 Number of tasks (procedures)
   - 3.5 ID of the entry task (procedure)
   - 3.6 Number of interprogram or external inputs
   - 3.7 Number of interprogram or external outputs
   - 3.8 For each interprogram or external input used by the program:
     - 3.8.1 Input ID and type code (interprogram or external)
     - 3.8.2 Number of the consumer tasks in the program
     - 3.8.3 IDs of all consumer tasks
   - 3.9 For each interprogram or external output produced by the program:
     - 3.9.1 Output ID and type code (interprogram or external)
     - 3.9.2 ID of the producer task
   - 3.10 Number of intraprogram-intertask input/output variables used within the program
   - 3.11 For each intraprogram-intertask input/output variable:
     - 3.11.1 ID/name of the input/output variable
     - 3.11.2 Type code
     - 3.11.3 ID of the producer task
     - 3.11.4 Size (= number of bits)
     - 3.11.5 Production rate
4. For each task (procedure) in every program:
   - 4.1 Task (procedure) ID/name
   - 4.2 Task enumeration number within the program
   - 4.3 Permanent memory (= number of 16-bit words)
   - 4.4 Temporary memory (= number of 16-bit words)
   - 4.5 Minimum and maximum numbers of repetitions (loops) per one execution of the program
   - 4.6 Processing load per repetition
     - 4.6.1 Single precision operations
       - Number of additions/subtractions
       - Number of multiplications
       - Number of divisions
     - 4.6.2 Double precision operations.
       - Number of additions/subtractions
       - Number of multiplications
       - Number of divisions
   - 4.7 Number of logical Boolean operations

4.8 Average number of control operations:
- Per one arithmetic operation
- Per one logical operation

4.9 Calls of single precision standard subroutines:
- Square root
- Sine/Cosine
- Arctan
- Exponential
- Logarithm
- Arc sin/Arc cos

4.10 Calls of double precision standard subroutines:
- Square root
- Sine/Cosine
- Arctan
- Exponential
- Logarithm
- Arc sin/Arc cos

4.11 Number of nonstandard tasks (procedures) called

4.12 For each nonstandard task (procedure) called:
- Task ID
- Number of calls

4.13 Number of successor tasks (procedures)

4.14 For each successor task (procedure):
- ID of the successor task (procedure)
- Probability or logical condition of transition

4.15 Number of intraprogram-intertask inputs used

4.16 IDs of all intraprogram-intertask inputs

4.17 Number of intraprogram-intertask outputs used

4.18 IDs of all intraprogram-intertask outputs

5. Number of interprogram inputs/outputs in the process model

6. For each interprogram input/output:
   - 6.1 ID/name of the input/output
   - 6.2 Type code
   - 6.3 ID of the producer program
   - 6.4 Size (= number of bits)
   - 6.5 Production rate

7. Number of external inputs in the process model

8. For each external input:
   - 8.1 ID/name of the input
   - 8.2 Type code
   - 8.3 ID/name of the source (i.e., of the External I/O Handler program)
   - 8.4 Size (= number of bits)
   - 8.5 Production rate
   - 8.6 ID code of the connecting DP/M PE
   - 8.7 ID code of the connecting sensor

9. Number of external outputs

10. For each external output:
    - 10.1 ID/name of the output
    - 10.2 Type code

10.3 ID/name of the producer (i.e., of the External I/O Handler program)
10.4 Size (= number of bits)
10.5 Production rate
10.6 ID code of the connecting DP/M PE
10.7 ID code of the connecting actuator

FILE NAME: Report on Computer-Independent Model of the Process
DESCRIPTOR: Documentation of the computer-independent model of the process
FORM. Computer-printed report
CONTENTS:
This report is a printed version of and has the same contents as the inputted Computer-independent Model of the Process (card images constituting a disk file). In addition, this report shall contain the maximally-parallel predecessor-successor graph and a companion program graph for each program in the process model.

FILE NAME: Computer Hardware Definition (Input File)
DESCRIPTOR: Exogenous input to Step 2 of the Process Construction Procedure
FORM Card images constituting a disk file
CONTENTS:

1. Computer ID
2. Computer Description
3. Memory size (= number of 16-bit words)
4. Speed of single precision operations (minimum and maximum execution times):
   - 4.1 Addition/subtraction
   - 4.2 Multiplication
   - 4.3 Division
5. Speed of double precision operations (minimum and maximum execution times):
   - 5.1 Addition/subtraction
   - 5.2 Multiplication
   - 5.3 Division
6. Logical/Boolean operations (average time)
7. Control operations (average time)
8. Subroutine Linkage (average time)
9. Average execution times for standard single precision subroutines
   - 9.1 Square root
   - 9.2 Sine/Cosine
   - 9.3 Arctan
   - 9.4 Exponential
   - 9.5 Logarithm
   - 9.6 Arc sin/Arc cos
10. Average execution times for standard double precision subroutines
    - 10.1 Square root
    - 10.2 Sine/Cosine
    - 10.3 Arctan
    - 10.4 Exponential
    - 10.5 Logarithm
    - 10.6 Arc sin/Arc cos
11. Memory Allocation guidelines for
    - 11.1 Single precision arithmetic operations:
      - Number of permanent memory words per operation
      - Number of temporary memory words per operation
    - 11.2 Double precision arithmetic operations:
      - Number of permanent memory words per operation
      - Number of temporary memory words per operation
    - 11.3 Subroutine linkages, logical operations, and control operations:
      - Number of permanent memory words
      - Number of temporary memory words
12. Bus capacity

FILE NAME: Report on Computer Hardware Definition
DESCRIPTOR: Documentation of the computer hardware model for which the process is to be constructed
FORM: Computer-printed report
CONTENTS:
This report is a printed version of and has the same contents as the Computer Hardware Definition input file.

FILE NAME: I/O Control Data File

DESCRIPTOR: Hardware-assignment dependent data (produced in Step 7 of the Process Construction Procedure) for controlling (routing) the interprocessor and external I/O

FORM: Disk file compatible with simulator inputs

CONTENTS:

The contents and organization of this file are dependent on the design of the traffic control subsystem within the DP/M Executive(s). Therefore, this file cannot be specified now. However, its contents shall be similar to that of the companion Report on I/O Message Routing, which is specified next in this document.

FILE NAME: Report on I/O Message Routing

DESCRIPTOR: A hardware-assignment-dependent report (produced in Step 7 of the Process Construction Procedure) defining the I/O message traffic in the process; its main function is system documentation

FORM: Computer-printed report

CONTENTS:

1. Number of interprogram I/O messages
2. Number of external I/O messages
3. For each interprogram I/O message:
   - 3.1 ID/name
   - 3.2 Type code
   - 3.3 Size (= number of bits)
   - 3.4 ID of the producer program and its PE assignment
   - 3.5 Number of consumer programs
   - 3.6 For each consumer program:
     - 3.6.1 Program ID/name
     - 3.6.2 Program PE assignment
   - 3.7 Production Rate
4. For each external I/O message:
   - 4.1 ID/name
   - 4.2 Type code
   - 4.3 ID/name of the consumer/producer (i.e., of the External I/O Handler program)
   - 4.4 Size (= number of bits)
   - 4.5 Production rate
   - 4.6 ID code of the connecting DP/M PE
5. Routing list for all interprogram I/O messages (ordered by increasing I/O message IDs), where each entry contains:
   - 5.1 Message ID/Name
   - 5.2 Type code
   - 5.3 Size (= number of bits)
   - 5.4 Production rate
   - 5.5 ID/name of the producing program
   - 5.6 PE assignment of the producing program
   - 5.7 Number of destination PEs
   - 5.8 IDs of all destination PEs
6. Entry/exit points of all external I/O messages (ordered by increasing I/O message IDs), where each entry contains:
   - 6.1 Message ID/Name
   - 6.2 Type code
   - 6.3 Size (= number of bits)
   - 6.4 Production rate
   - 6.5 ID/name of the consuming/producing External I/O Handler (program)
   - 6.6 ID code of the connecting DP/M PE
   - 6.7 ID code of the connecting sensor/actuator

FILE NAME: Program Execution Control Data File

DESCRIPTOR: Hardware-assignment-dependent data (produced in Step 8 of the Process Construction Procedure) to be used by the DP/M Executive to control program execution

FORM: Disk file compatible with simulator inputs

CONTENTS:

The contents and organization of this file are dependent on the design of the DP/M Executive(s); hence, this file cannot be specified here. For a description of the program execution control data used by the current version of the DP/M Executive, refer to the section describing the design of the DP/M Executive.

FILE NAME: Report on Program Execution Control

DESCRIPTOR: A hardware-assignment-dependent report (produced in Step 8 of the Process Construction Procedure) documenting the program execution control data to be used by the DP/M Executive

FORM: Computer-printed report

CONTENTS:

The content of this report is dependent on the design of the DP/M Executive subsystem, hence, it is not specified here. For a description of the program execution control data used by the current version of the DP/M Executive, refer to the section describing the design of the DP/M Executive.

FILE NAME: Memory Map and System Configuration File
DESCRIPTOR: Endogenous output of Step 9 of the Process Construction Procedure to be used by processor simulators
FORM: Disk file compatible with simulator inputs
CONTENT:

1. Process Model ID/descriptor
2. Number of Processing Elements in the DP/M configuration
3. For each Processing Element:
   - 3.1 Number of external software modules (i.e., those not nested in other software modules) - such as tasks (procedures), standard or special subroutines - to be loaded into the memory of the PE under consideration
   - 3.2 Number of data sets (areas, buffer areas files) to reside in the memory of the PE under consideration
   - 3.3 For each external software module:
     - 3.3.1 Module ID/name
     - 3.3.2 Loading address
     - 3.3.3 Size (= number of 16-bit words)
   - 3.4 For each data set:
     - 3.4.1 Data set ID/name
     - 3.4.2 Loading address
     - 3.4.3 Size (= number of 16-bit words)
4. Master directory of all software modules in the process module, each entry containing:
   - 4.1 Module ID/name
   - 4.2 Type code (designating whether a standard subroutine, special subprogram, task of a program, etc.)
   - 4.3 Size (= number of 16-bit words)
   - 4.4 PE assignment
   - 4.5 Configuration control/functional descriptor
5. Master directory of all data sets in the process model, each entry containing:
   - 5.1 Data set ID/name
   - 5.2 Type code
   - 5.3 Size (= number of 16-bit words)
   - 5.4 PE assignment
   - 5.5 Configuration control/functional descriptor

| | |
|---|---|
| FILE NAME: | Report on Memory Maps and System Configuration |
| DESCRIPTOR: | Endogenous output of Step 9 of the Process Construction Procedure to be used by the process designer and for process documentation purposes |
| FORM: | A computer-printed report |
| CONTENTS: | Same as the Memory Map and System Configuration File |

FILE NAME: Report on the Correctness of Program Structure
DESCRIPTOR: Output of Step 4 of the Process Construction Procedure, which is to be used mainly by the process designer as a diagnostic document
FORM Computer-printed report
CONTENTS:

1. Process ID/name
2. Number of programs in the process model
3. For each program in the process.
   - 3.1 A summary description of the input program model containing:
     - 3.3.1 Program ID/name and type code
     - 3.3.2 Number of tasks (procedures)
     - 3.3.3 ID of the entry task (procedure)
     - 3.3.4 For each task procedure in the program.
       - (a) Task (represented by a node in the program graph) ID/name
       - (b) Task ID number assigned by the topological ordering sort (to be called topological ID number)
       - (c) Number of links pointing to the task (node), which is equal to the number predecessor tasks
       - (d) Number of links emanating from the task (node), which is equal to the number of successor tasks
       - (e) ID/name of each successor task (node) and the corresponding transition probability or the logical branching condition
     - 3.3.5 Transition Probability (or logical branching condition) Matrix
     - 3.3.6 Node Adjacency Matrix
     - 3.3.7 Path Matrix
     - 3.3.8 List of all execution paths through the program graph, each path being defined by an ordered list of task IDs
     - 3.3.9 A summary statement on the correctness of program structure, including the information on whether the designated entry and exit tasks (nodes) actually are such and whether there are no loops through the tasks.

FILE NAME Report on Program Synthetic Models

DESCRIPTOR. Output of Step 4 of the Process Construction Procedure, which is to be used mainly by the process designer to decide whether the current decomposition of the process into programs is acceptable

FORM: Computer-printed report

CONTENTS:

1. Process ID/name
2. Number of programs in the process model
3. For each program in the process model:
   - 3.1 Program ID/name
   - 3.2 Scheduling parameters and iteration rate
   - 3.3 Number of tasks (nodes)
   - 3.4 ID of the entry task (node)
   - 3.5 For each task (node)
     - 3.5.1 Task ID/Name
     - 3.5.2 Permanent memory (= number of 16-bit words)
     - 3.5.3 Temporary memory (= number of 16-bit words)
     - 3.5.4 Estimate of the minimum run time
     - 3.5.5 Estimate of the maximum run time
     - 3.5.6 Minimum and maximum number of repetitions
     - 3.5.7 Indegree (= number of predecessor tasks)
     - 3.5.8 Outdegree (= number of successor tasks)
     - 3.5.9 For each successor task, the task ID and the transition probability (or the logical branching condition) to that task
     - 3.5.10 Number of intraprogram-intertask inputs used
     - 3.5.11 IDs of all intraprogram-intertask inputs
   - 3.6 Number of alternative execution paths through the program graph
   - 3.7 For each execution path, an ordered task (node) list and the probability (or the logical condition) of the path
   - 3.8 The worst case processor loading per one unit of real-time by the program
   - 3.9 Memory loading by the program, divided into
     - (a) Permanent memory requirements for tasks, data sets, and the called subroutines
     - (b) Temporary memory
   - 3.10 Number of standard and special subroutines called by the program
   - 3.11 List of the called standard or special subroutines
   - 3.12 The maximally parallel predecessor-successor graph
   - 3.13 Number of interprogram or external inputs
   - 3.14 Number of interprogram or external outputs
   - 3.15 For each interprogram or external input:
     - 3.15.1 Input ID/name and type code
     - 3.15.2 Size (= number of bits)
     - 3.15.3 Number of consumer tasks in the program
     - 3.15.4 IDs of the consumer tasks
     - 3.15.5 Arrival rate

3.16 For each interprogram or external output:
- 3.16.1 Output ID/name and type code
- 3.16.2 Size (= number of bits)
- 3.16.3 ID of the producer task
- 3.16.4 Production rate

3.17 Number of intraprogram-intertask input/output variables used within the program

3.18 For each intraprogram-intertask input/output variable:
- 3.18.1 ID/name and type code
- 3.18.2 Size (= number of 16-bit words)
- 3.18.3 ID of the producer task
- 3.18.4 Production rate

FILE NAME: Report on Hardware Configuration and Assignments

DESCRIPTOR: Exogenous output of the Hardware Resource Allocation Algorithm: (Step 6 of Process Construction Procedure) also to be used as a diagnostic report by the process designer

FORM: Computer-printed report

CONTENTS:

1. The required number of Processing Elements
2. Projected total loading of the bus system (the term "total" here implies that no separation of the bus load into the Local and Global bus traffic components is made).
3. For each required Processing Element:
   - 3.1 Number of software modules assigned
   - 3.2 Number of data sets assigned
   - 3.3 List of all assigned software modules, each entry containing:
     - 3.3.1 ID/name of the software module
     - 3.3.2 Type code
     - 3.3.3 Permanent memory required
     - 3.3.4 Temporary memory required
     - 3.3.5 The contribution of the module to the loading of the processor time per one unit of real-time
     - 3.3.6 The contribution of the module to the bus traffic
   - 3.4 List of all data sets that require memory allocation
     - 3.4.1 ID/name of that data set
     - 3.4.2 Type code
     - 3.4.3 Size (= number of 16-bit words)
   - 3.5 Projected total loading of the processor time per one unit of real-time
   - 3.6 Total loading of the permanent memory (= number of 16-bit words)
   - 3.7 Total loading of the temporary memory (= number of 16-bit words)
4. List of all software modules in the process model, each entry containing:
   - 4.1 ID/name of the module
   - 4.2 Type code
   - 4.3 PE assignment
5. List of all data sets in the process model, each entry containing:
   - 5.1 ID/name of the data set
   - 5.2 Type code
   - 5.3 PE assignment

## APPENDIX B

## PROCESS CONSTRUCTION CASE STUDY
## SIMULATION REPORT

**********

PROCESSING ELEMENT CONNECTIVITY SPECIFICATION

**********

| GLOBAL PLS CONNECTIVITY | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 11 | 12 | 13 | 14 | 15 | | | | | |

| | | |
|---|---|---|
| AFFINITY GROUP NUMBER 1 | CONNECTIVITY | 1 |
| AFFINITY GROUP NUMBER 2 | CONNECTIVITY | 2 |
| AFFINITY GROUP NUMBER 3 | CONNECTIVITY | 3 |
| AFFINITY GROUP NUMBER 4 | CONNECTIVITY | 4 |
| AFFINITY GROUP NUMBER 5 | CONNECTIVITY | 5 |
| AFFINITY GROUP NUMBER 6 | CONNECTIVITY | 6 |
| AFFINITY GROUP NUMBER 7 | CONNECTIVITY | 7 |
| AFFINITY GROUP NUMBER 8 | CONNECTIVITY | 8 |
| AFFINITY GROUP NUMBER 9 | CONNECTIVITY | 9 10 |
| AFFINITY GROUP NUMBER 10 | CONNECTIVITY | 11 |
| AFFINITY GROUP NUMBER 11 | CONNECTIVITY | 12 |
| AFFINITY GROUP NUMBER 12 | CONNECTIVITY | 13 |
| AFFINITY GROUP NUMBER 13 | CONNECTIVITY | 14 |
| AFFINITY GROUP NUMBER 14 | CONNECTIVITY | 15 |

**********

Pages 482, 483 and 484 are Blank.

AVIONIC TASKS TO PROCESSING ELEMENT ASSIGNMENT

**********

PROCESSING ELEMENT 1

TASKS ASSIGNED 1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20

PROCESSING ELEMENT 2

TASKS ASSIGNED 41 49 50 55

PROCESSING ELEMENT 3

TASKS ASSIGNED 43 44

PROCESSING ELEMENT 4

TASKS ASSIGNED 45 46

PROCESSING ELEMENT 5

TASKS ASSIGNED 47

PROCESSING ELEMENT 6

TASKS ASSIGNED 51

PROCESSING ELEMENT 7

TASKS ASSIGNED 52 56 61

PROCESSING ELEMENT 8

TASKS ASSIGNED 53 54

PROCESSING ELEMENT 9

TASKS ASSIGNED 57 157

PROCESSING ELEMENT 10

TASKS ASSIGNED 58

PROCESSING ELEMENT 11

TASKS ASSIGNED 59

PROCESSING ELEMENT 12

TASKS ASSIGNED 63

PROCESSING ELEMENT 13

TASKS ASSIGNED 65

PROCESSING ELEMENT 14

TASKS ASSIGNED 67

PROCESSING ELEMENT 15

TASKS ASSIGNED 69

```
DPN SIMULATION TEST CASE                                    17:00:39    12/19/74
              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 1          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                      REPORT STOP TIME =      0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED  =            0
TOTAL INTERRUPT SERVICE TIME   =      0.0                        PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME     =      0.021EC3                   PERCENT =   21.80
TOTAL APPLICATIONS PROGRAM TIME =     0.0                        PERCENT =    0.0
TOTAL IDLE TIME                =      0.07H157                   PERCENT =   78.20
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                            .
AVAILABLE MEMORY =       4096                                                  .
MAXIMUM USED     =       1640                    AVERAGE USED =       1650
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                           .
NUMBER OF INCOMING MESSAGES =          44    INCOMING MESSAGES MISSED =        0
NUMBER OF OUTGOING MESSAGES =          49    OUTGOING MESSAGES MISSED =        0
*******************************************************************************

DPN SIMULATION TEST CASE                                    17:00:39    12/19/74
.             COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 2          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                      REPORT STOP TIME =      0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED  =            0                                 .
TOTAL INTERRUPT SERVICE TIME   =      0.0                        PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME     =      0.004172                   PERCENT =    4.17
TOTAL APPLICATIONS PROGRAM TIME =     0.011440                   PERCENT =   11.44
TOTAL IDLE TIME                =      0.084380                   PERCENT =   84.39
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                            .
AVAILABLE MEMORY =       4096                                                  .
MAXIMUM USED     =       3995                    AVERAGE USED =       3995
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                           .
NUMBER OF INCOMING MESSAGES =          21    INCOMING MESSAGES MISSED =        0
NUMBER OF OUTGOING MESSAGES =          14    OUTGOING MESSAGES MISSED =        0
*******************************************************************************
```

```
DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 3          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                        REPORT STOP TIME =     0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                               .
NUMBER OF INTERRUPTS SERVICED   =          0                                   .
TOTAL INTERRUPT SERVICE TIME    =     0.0                        PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =     0.000078                   PERCENT =    0.88
TOTAL APPLICATIONS PROGRAM TIME =     0.022000                   PERCENT =   22.68
TOTAL IDLE TIME                 =     0.076442                   PERCENT =   76.44
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                             .
AVAILABLE MEMORY =      4096                                                   .
MAXIMUM USED     =      2238                          AVERAGE USED =      2238
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                            .
NUMBER OF INCOMING MESSAGES =         2     INCOMING MESSAGES MISSED =         0
NUMBER OF OUTGOING MESSAGES =         6     OUTGOING MESSAGES MISSED =         0
*******************************************************************************

DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 4          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                        REPORT STOP TIME =     0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                               .
NUMBER OF INTERRUPTS SERVICED   =          0                                   .
TOTAL INTERRUPT SERVICE TIME    =     0.0                        PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =     0.001239                   PERCENT =    1.24
TOTAL APPLICATIONS PROGRAM TIME =     0.001630                   PERCENT =    1.63
TOTAL IDLE TIME                 =     0.097131                   PERCENT =   97.13
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                             .
AVAILABLE MEMORY =      4096                                                   .
MAXIMUM USED     =      3437                          AVERAGE USED =      3437
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                            .
NUMBER OF INCOMING MESSAGES =        13     INCOMING MESSAGES MISSED =         0
NUMBER OF OUTGOING MESSAGES =         1     OUTGOING MESSAGES MISSED =         0
*******************************************************************************
```

```
DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 5          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =     0.0                    REPORT STOP TIME =     0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED   =          0                                  .
TOTAL INTERRUPT SERVICE TIME    =    0.0          PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =    0.002683     PERCENT =    2.68
TOTAL APPLICATIONS PROGRAM TIME =    0.037842     PERCENT =   37.84
TOTAL IDLE TIME                 =    0.059475     PERCENT =   59.47
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                            .
AVAILABLE MEMORY =     4096                                                   .
MAXIMUM USED     =     3019                        AVERAGE USED =     3019
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                           .
NUMBER OF INCOMING MESSAGES =        7     INCOMING MESSAGES MISSED =        0
NUMBER OF OUTGOING MESSAGES =        6     OUTGOING MESSAGES MISSED =        0
*******************************************************************************

DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 6          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =     0.0                    REPORT STOP TIME =     0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED   =          0                                  .
TOTAL INTERRUPT SERVICE TIME    =    0.0          PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =    0.001598     PERCENT =    1.59
TOTAL APPLICATIONS PROGRAM TIME =    0.010956     PERCENT =   10.96
TOTAL IDLE TIME                 =    0.087456     PERCENT =   87.46
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                            .
AVAILABLE MEMORY =     4096                                                   .
MAXIMUM USED     =     2130                        AVERAGE USED =     2130
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                           .
NUMBER OF INCOMING MESSAGES =        6     INCOMING MESSAGES MISSED =        0
NUMBER OF OUTGOING MESSAGES =        3     OUTGOING MESSAGES MISSED =        0
*******************************************************************************
```

```
DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 7              PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                      REPORT STOP TIME =      0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                                .
NUMBER OF INTERRUPTS SERVICED   =            0                                  .
TOTAL INTERRUPT SERVICE TIME    =      0.0                      PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =      0.003984                 PERCENT =   3.98
TOTAL APPLICATIONS PROGRAM TIME =      0.012456                 PERCENT =  12.46
TOTAL IDLE TIME                 =      0.083560                 PERCENT =  83.56
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                              .
AVAILABLE MEMORY =      4096                                                    .
MAXIMUM USED     =      2190                        AVERAGE USED =       2190
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                             .
NUMBER OF INCOMING MESSAGES =          14    INCOMING MESSAGES MISSED =          0
NUMBER OF OUTGOING MESSAGES =          12    OUTGOING MESSAGES MISSED =          0
```

```
DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 8              PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                      REPORT STOP TIME =      0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                                .
NUMBER OF INTERRUPTS SERVICED   =            0                                  .
TOTAL INTERRUPT SERVICE TIME    =      0.0                      PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =      0.002055                 PERCENT =   2.05
TOTAL APPLICATIONS PROGRAM TIME =      0.015190                 PERCENT =  15.19
TOTAL IDLE TIME                 =      0.082755                 PERCENT =  82.75
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                              .
AVAILABLE MEMORY =      4096                                                    .
MAXIMUM USED     =      1990                        AVERAGE USED =       1990
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                             .
NUMBER OF INCOMING MESSAGES =           9    INCOMING MESSAGES MISSED =          0
NUMBER OF OUTGOING MESSAGES =           9    OUTGOING MESSAGES MISSED =          0
```

```
DPP SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 9              PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                      REPORT STOP TIME =      0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                               .
NUMBER OF INTERRUPTS SERVICED   =            0                                 .
TOTAL INTERRUPT SERVICE TIME    =      0.0                    PERCENT =     0.0
TOTAL SYSTEMS PROGRAM TIME      =      0.001373               PERCENT =     1.37
TOTAL APPLICATIONS PROGRAM TIME =      0.009956               PERCENT =     9.96
TOTAL IDLE TIME                 =      0.088671               PERCENT =    88.67
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                             .
AVAILABLE MEMORY =       4096                                                  .
MAXIMUM USED     =       3922                    AVERAGE USED =       3822
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                            .
NUMBER OF INCOMING MESSAGES =          5     INCOMING MESSAGES MISSED =          0
NUMBER OF OUTGOING MESSAGES =          5     OUTGOING MESSAGES MISSED =          0
*******************************************************************************

DPP SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR10              PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                      REPORT STOP TIME =      0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                               .
NUMBER OF INTERRUPTS SERVICED   =            0                                 .
TOTAL INTERRUPT SERVICE TIME    =      0.0                    PERCENT =     0.0
TOTAL SYSTEMS PROGRAM TIME      =      0.000725               PERCENT =     0.72
TOTAL APPLICATIONS PROGRAM TIME =      0.003024               PERCENT =     3.02
TOTAL IDLE TIME                 =      0.096251               PERCENT =    96.25
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                             .
AVAILABLE MEMORY =       4096                                                  .
MAXIMUM USED     =       3926                    AVERAGE USED =       3926
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                            .
NUMBER OF INCOMING MESSAGES =          3     INCOMING MESSAGES MISSED =          0
NUMBER OF OUTGOING MESSAGES =          2     OUTGOING MESSAGES MISSED =          0
*******************************************************************************
```

```
DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR11          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =     0.0                        REPORT STOP TIME =     0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED   =            0                                .
TOTAL INTERRUPT SERVICE TIME    =     0.0                      PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =     0.001766                 PERCENT =   1.79
TOTAL APPLICATIONS PROGRAM TIME =     0.036484                 PERCENT =  36.48
TOTAL IDLE TIME                 =     0.061730                 PERCENT =  61.73
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                            .
AVAILABLE MEMORY =      4096                                                  .
MAXIMUM USED     =      3419                       AVERAGE USED =      3419
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                           .
NUMBER OF INCOMING MESSAGES =         6     INCOMING MESSAGES MISSED =        0
NUMBER OF OUTGOING MESSAGES =        27     OUTGOING MESSAGES MISSED =        0
*******************************************************************************
```

```
DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR12          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =     0.0                        REPORT STOP TIME =     0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED   =            0                                .
TOTAL INTERRUPT SERVICE TIME    =     0.0                      PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =     0.002244                 PERCENT =   2.24
TOTAL APPLICATIONS PROGRAM TIME =     0.040968                 PERCENT =  40.97
TOTAL IDLE TIME                 =     0.056788                 PERCENT =  56.79
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                            .
AVAILABLE MEMORY =      4096                                                  .
MAXIMUM USED     =      3515                       AVERAGE USED =      3515
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                           .
NUMBER OF INCOMING MESSAGES =        10     INCOMING MESSAGES MISSED =        0
NUMBER OF OUTGOING MESSAGES =         6     OUTGOING MESSAGES MISSED =        0
*******************************************************************************
```

```
DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR13          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                    REPORT STOP TIME =      0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED   =           0                                 .
TOTAL INTERRUPT SERVICE TIME    =     0.0                      PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =     0.001750                 PERCENT =   1.75
TOTAL APPLICATIONS PROGRAM TIME =     0.046260                 PERCENT =  46.26
TOTAL IDLE TIME                 =     0.051990                 PERCENT =  51.99
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                            .
AVAILABLE MEMORY =      4096                                                  .
MAXIMUM USED     =      1656                        AVERAGE USED =      1656
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                           .
NUMBER OF INCOMING MESSAGES =          8     INCOMING MESSAGES MISSED =       0
NUMBER OF OUTGOING MESSAGES =          4     OUTGOING MESSAGES MISSED =       0
*******************************************************************************

DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR14          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                    REPORT STOP TIME =      0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED   =           0                                 .
TOTAL INTERRUPT SERVICE TIME    =     0.0                      PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =     0.002694                 PERCENT =   2.69
TOTAL APPLICATIONS PROGRAM TIME =     0.022122                 PERCENT =  22.12
TOTAL IDLE TIME                 =     0.075184                 PERCENT =  75.18
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                            .
AVAILABLE MEMORY =      4096                                                  .
MAXIMUM USED     =      3139                        AVERAGE USED =      3139
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                           .
NUMBER OF INCOMING MESSAGES =         15     INCOMING MESSAGES MISSED =       0
NUMBER OF OUTGOING MESSAGES =          6     OUTGOING MESSAGES MISSED =       0
*******************************************************************************
```

```
DPM SIMULATION TEST CASE                                    17:00:39    12/19/74
.          COMPUTER UTILIZATION SUMMARY FOR PROCESSOR15          PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =      0.0                    REPORT STOP TIME =      0.100000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                              .
NUMBER OF INTERRUPTS SERVICED    =            0                               .
TOTAL INTERRUPT SERVICE TIME     =     0.0                      PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME       =     0.003504                 PERCENT =   3.50
TOTAL APPLICATIONS PROGRAM TIME  =     0.052407                 PERCENT =  52.41
TOTAL IDLE TIME                  =     0.044089                 PERCENT =  44.09
-------------------------------------------------------------------------------
.                         MEMORY UTILIZATION DATA                             .
AVAILABLE MEMORY =      4096                                                  .
MAXIMUM USED     =      4070                          AVERAGE USED =      4070
-------------------------------------------------------------------------------
.                         MESSAGE UTILIZATION DATA                            .
NUMBER OF INCOMING MESSAGES =        20     INCOMING MESSAGES MISSED =        0
NUMBER OF OUTGOING MESSAGES =        12     OUTGOING MESSAGES MISSED =        0
*******************************************************************************

*******************************************************************************
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS    9
-------------------------------------------------------------------------------
TOTAL TRAFFIC                =       0.84 KBPS
TRAFFIC UTILIZED FOR DATA    =       0.34 KBPS  40.48 PERCENT
TRAFFIC UTILIZED FOR HEADER  =       0.34 KBPS  40.48 PERCENT
TRAFFIC UTILIZED FOR SYNC    =       0.06 KBPS   7.14 PERCENT
TRAFFIC UTILIZED FOR GAP     =       0.10 KBPS  11.90 PERCENT
*******************************************************************************
*******************************************************************************
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS   10
-------------------------------------------------------------------------------
TOTAL TRAFFIC                =       0.0  KBPS
*******************************************************************************
*******************************************************************************
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS   11
-------------------------------------------------------------------------------
TOTAL TRAFFIC                =       0.0  KBPS
*******************************************************************************
*******************************************************************************
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS   12
-------------------------------------------------------------------------------
TOTAL TRAFFIC                =       0.0  KBPS
*******************************************************************************
```

```
**********
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS   13
----------
TOTAL TRAFFIC                 =       0.0  KBPS
**********
**********
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS   14
----------
TOTAL TRAFFIC                 =       0.0  KBPS
**********
**********
                BUS TRAFFIC DECOMPOSITION FOR GLOBAL BUS   99
----------
TOTAL TRAFFIC                 =      84.67 KBPS
TRAFFIC UTILIZED FOR DATA     =      46.92 KBPS  55.42 PERCENT
TRAFFIC UTILIZED FOR HEADER   =      25.67 KBPS  30.32 PERCENT
TRAFFIC UTILIZED FOR SYNC     =       4.53 KBPS   5.35 PERCENT
TRAFFIC UTILIZED FOR GAP      =       7.55 KBPS   8.92 PERCENT
**********
```

## BIBLIOGRAPHY


Booton, W.C.; Daggett, D.H.; et al.; *A Conceptual Definition Study for a Digital Avionics Information System (Approach I)*, AFAL-TR-73-300, Volumes I, II, and III, General Dynamics/Fort Worth, October 1973.

Brodnax, Dr. C.T.; *A Conceptual Study for a Digital Avionics Information System (Approach II)*, AFAL-TR-73-427, Volumes I and II, Texas Instruments Incorporated, January 1974.

Bucy, J.F.; "World Price Leadership," *Vectors Pamphlet*, Texas Instruments Incorporated, 1974.

Chamberlin, L.A., et al.; *Design of the CORE Elements of the Digital Avionics Information System*, AFAL-TR-74-245, Volumes I, II, and III, Texas Instruments Incorporated, 18 November 1974.

Griffeth, Dr. V.V.; Keifer, L.F., Jr.; Paxhia, E.C.; et al.; *Aircraft Avionics Trade-Off Study (AATOS)*, ASD/XR 73-20, McDonnell Aircraft Company, November 1973.

Gordon, Geoffrey, *System Simulation*, Prentice Hall, 1969.

Cross, J.P.; Hoffman, M.H.; *The Application of Information Transfer Techniques for Solving the Internal Communication Requirements of a Night AX Aircraft*, AFAL-TR-73-226, Volume I, SCI Systems Inc., April 1973.

Johnson, M.D., et al.; *All Semiconductor Distributed Aerospace Processor/Memory Study*, AFAL-TR-73-226, Volume II, Honeywell Inc., August 1973.

Kilpatrick, P.S.; Marshall, W.C.; Scales, E.A.; *All Semiconductor Distributed Aerospace Processor/Memory Study*, AFAL-TR-72-226, Volume I, Honeywell Inc., August 1973.

Murphy, L.R.; Avizienis, A.A.; Rennels, A.A.; et al.; *Fault Tolerant Avionic Systems Architectures Study*, AFAL-TR-74-102, Ultrasystems Inc., June 1974.

Thurber, K.J.; Jensen, E.D.; Jack, L.A., "A Systematic Approach to the Design of Digital Bussing Structures," *Proceedings of the FJCC, 1972*, AFIPS Press, pp. 719–740.

*Military Specification-Electronic Equipment, Airborne, General Specification for*, MIL-E-5400P, 2 July 1973.

*Military Standard-Aircraft Internal Time Division Multiplex Data Bus*, MIL-STD-1553, 30 August 1973.

*Military Standard-Electrical Power, Aircraft, Characteristics and Utilization of*, MIL-STD-704A, 5 May 1970.

*Report of the USAF Scientific Advisory Board Ad Hoc Committee to Review On-Board Test and Recording Systems*, 13 August 1973.

*User's Manual for the Basic Simulator*, Texas Instruments Incorporated, February 1975.

*User's Manual for the DP/M Processing Element Simulation System*, Texas Instruments Incorporated, February 1975.

*User's Manual for the DP/M System Network Simulation System*, Texas Instruments Incorporated, February 1975.


**Preceding page blank**

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ac | Alternating Current |
| AG | Affinity Group |
| AGE | Aerospace Ground Equipment |
| ALU | Arithmetic Logic Unit |
| AND | Logical And Operation |
| ANS | American National Standard |
| ARINC | Aeronautical Radio, Inc. |
| ATC | Autonomous Transfer Channel |
| ATCL | Autonomous Transfer Control Logic |
| ATCR | Autonomous Transfer Control Register |
| | |
| BC | Branch On Condition Long Instruction |
| BCD | Binary Coded Decimal |
| BCS | Branch On Condition Short Instruction |
| BDWT | Bus Dominance Watchdog Timer |
| BGR | Bus Grant Signal |
| BILR | Branch Indirect And Link Register Instruction |
| BILU | Bus Interface Logic Unit |
| BIT | Built-In-Test |
| BITU | Bus Interface Translation Unit |
| BIU | Bus Interface Unit |
| BMI | Bus Master Interface |
| BPC | Basic Process Constructor |
| BQWT | Bus Quiescence Watchdog Timer |
| BREL | Bus Release Signal |
| BRQ | Bus Request Signal |
| BSI | Bus Slave Interface |
| BSSCAN | Branch Successor Scanner |
| | |
| C | Constant Instruction Modification |
| CAS | Close Air Support |
| CAW | Command Address Word |
| CCD | Charge-Coupled Device |
| CDMG | Computer-Dependent Model Generator |
| CIMG | Computer-Independent Model Generator |
| $C^2L$ | Current-Coupled Logic |
| CLR | Clear/Reset Signal |
| CMOS | Complementary Metal Oxide Semiconductor |
| CPC | Current Position Of Control |
| CPU | Central Processing Unit |

Preceding page blank

| | |
|---|---|
| D | Direct Instruction Modification |
| DAIS | Digital Avionics Information System |
| dc | Direct Current |
| DIP | Dual In-Line Package |
| DISPIN | Scheduler Service Module |
| DMA | Direct Memory Access |
| DP/M | Distributed Processor/Memory |
| DTL | Diode Transistor Logic |
| DX | Direct Indexed Instruction Modification |
| | |
| ECDG | Execution Control Data Generator |
| ECL | Emitter-Coupled Logic |
| ECM | Electronic Countermeasures |
| EFL | Emitter-Follower Logic |
| EIA | Electronics Industries Association |
| | |
| FBW | Fly-by-Wire |
| FCLK | Free-Running Clock Signal |
| FEL | Future Event List |
| FIFO | First-In/First-Out |
| | |
| GBPR | Global Bus Position Register |
| GEX | Global Executive |
| GEXSCHED | Global Executive Scheduler |
| G/L | Global/Local |
| | |
| HRAA | Hardware Resource Allocation Algorithm |
| HS-RAM | High Speed (Read/Write) Random Access Memory |
| Hz | Hertz (Cycles) |
| | |
| IAK | Interrupt Acknowledge Signal |
| IAR | Input Address Register |
| I-BUS | Internal PE Bus |
| IC | Integrated Circuit |
| ID | Identification, Identifier |
| IDR | Input Data Register |
| $I^2L$ | Integrated Injection Logic |
| ILR | Input Length Register |
| ILS | Instruction Level Simulation |
| IMQ | Input Message Queue |
| IMSCAN | Input Message Scanner |
| INHB | Interrupt Inhibit Signal |

| | |
|---|---|
| INTI | Interrupt Interface |
| IOCDG | I/O Control Data Generator |
| IODI | Input/Output Data Interface |
| IOIU | Input/Output Interface Unit |
| IOLU | Input/Output Logic Unit |
| IOSA | I/O and Storage Analyzer |
| IQP | Input Queue Pointer |
| IRQ | Interrupt Request Signal |
| | |
| K | Kilo (Thousand) |
| Kbps | Kilo Bits Per Second |
| KIPS | Thousand Instructions Per Second |
| | |
| LED | Light-Emitting Diode |
| LEX | Local Executive |
| LRU | Line Replaceable Unit |
| LSI | Large-Scale Integration |
| LS-RAM | Low-Speed (Read/Write) Random Access Memory |
| | |
| mA·hr | Milliampere Hour |
| Mbps | Megabits Per Second |
| MBVS | Message Buffer Vector Space |
| MCLK | Master Clock Signal |
| MIAA | Message Identity Associative Address |
| MIAMM | Message Input Associative Match Map |
| MNOS | Metal Nitride Oxide Semiconductor |
| MOS | Metal Oxide Semiconductor |
| ms | Millisecond |
| MSB | Most Significant Bit |
| MSI | Medium-Scale Integration |
| MSTP | Master Stop Signal |
| MTBF | Mean Time Between Failures |
| mW | Milliwatt |
| | |
| NARBS | Night Angle Rate Bombing System |
| NMOS | Negative Channel Metal Oxide Semiconductor |
| NPRZ | Number of Predecessors |
| NRZ | Non-Return to Zero |
| ns | Nanosecond |
| | |
| OAR | Output Address Register |
| OLR | Output Length Register |

| | |
|---|---|
| OMRQST | Output Message (Request) Interpreter Module |
| OR | Logical Or Operation |
| PAP | Process Analysis Program |
| PATS | Performance Assurance Test Software |
| PC | Program Counter |
| PDIS | Process Development Information System |
| PE | Processing Element |
| PIO | Pseudo Input/Output Instruction |
| PIT | Programmable Interval Timer |
| PLA | Programmable Logic Array |
| P/M | Processor/Memory |
| PMOS | Positive Channel Metal Oxide Semiconductor |
| R | Register Instruction Modification |
| RAM | (Read/Write) Random Access Memory |
| RBM | Redundant Bus Management |
| RBMU | Redundant Bus Management Unit |
| RF | Register File |
| RI | Register Indirect Instruction Modification |
| RIA | Register Indirect Autoincrement Instruction Modification |
| ROM | Read Only Memory |
| RR | Register-To-Register |
| RT | Real Time |
| SCA | Simulation Control Algorithm |
| SOS | Silicon On Sapphire |
| SOW | Statement of Work |
| SSI | Smal-Scale Integration |
| SW | Status Word |
| TACK | Transfer Acknowledge Signal |
| TCEP | Task Completion Entry Point |
| TDM | Time-Division-Multiplex |
| TRQ | Transfer Request Signal |
| TTL | Transistor-Transistor Logic |
| USAF | United States Air Force |
| V | Volt |
| VMD | Virtual Message Distributor |