AD-A053 345 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
DESIGN OF THE PROCESSOR FOR SOFTWARE COMPATIBLE AVIONIC COMPUTE--ETC(U)
DEC 77 F G THOUROT

UNCLASSIFIED AFIT/GCS/EE/77-10 NL

1 OF 3
AD
A053345

AFIT/GCS/EE/77-10

(1)

DESIGN OF THE PROCESSOR FOR SOFTWARE COMPATIBLE AVIONICS COMPUTER FAMILY

- THESIS -

AFIT/GCS/EE/77-10

Frederick G. Thourot
Capt. USAF

AFIT/GCS/EE/77-10

DESIGN OF THE PROCESSOR FOR SOFTWARE COMPATIBLE AVIONICS COMPUTER FAMILY

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Frederick G. Thourot, B.S.E.E.

Capt. USAF

Graduate Computer Science

December 1977

Preface

This report presents the design of a microprogrammable processor which is based upon bipolar bit slice technology. In designing this processor, I was forced to make many difficult decisions and to accept certain compromises in an attempt to balance considerations of speed, microprogramming flexibility, and hardware costs. I hope that the information presented in this report will help future designers with their decisions.

I would like to express my sincere appreciation to Capt J. B. Peterson, my thesis advisor, for his advice and direction in this effort. I would also like to thank my faculty advisor, Capt Pete Miller, for his personal guidance during my studies at AFIT. I have the highest regard for these two individuals.

I would also like to thank Dr. Michael and Dr. Moon of the Air Force Avionic Laboratory who sponsored this thesis and provided many hours of assistance.

Most of all I would like to thank my wife, Nancy, for her encouragement and understanding.

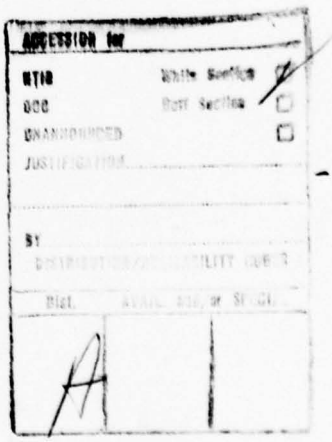


Table of Contents

	Page
Preface	ii
List of Figures	vii
List of Tables	ix
Abstract	xi
I. Introduction	1
Background	1
Problem	2
Objective and Scope	2
General Approach	3
Thesis Outline	3
II. Processor Requirements	5
Data Formats	5
Single Precision Integer	5
Double Precision Integer	6
Floating Point	6
Internal Registers	7
General Register File	7
Instruction Counter	8
Status Word	8
Interrupt Mask	8
Working Registers	8
Address Modes	9
Register Direct	9
Memory Direct	12
Memory Direct-Indexed	12
Memory Indirect	12
Memory Indirect with Pre-Indexing	12
Immediate Long	12
Immediate Long-Indexed	12
Immediate Short Positive	13
Immediate Short Negative	13
IC-Relative	13
Base Relative	13
Instruction Set	14
Interrupt System	14
Saving Old Computer State	16
New Computer State	18
Return from Interrupt	18
Interrupt Input/Output Instructions	18
Microcomputer I-BUS	19
Bus Master Control	21

Table of Contents

		Page
	Data Transfer and Control	23
	Master to Slave Send Cycle	23
	Master from Slave Receive Cycle	24
	General Bus Facilities	25
	Special Functions	25
	Input/Output	26
	DMA Channel	26
	Programmed Input/Output Channel	26
	Discrete Input/Output Channel	27
	Interval Timers	27
	Operational Speed Requirement	27
	Summary	29
III.	Design Overview	30
	Arithmetic and Logic Unit	30
	Computer Control Unit	32
	Interrupt Control Unit	32
	Input/Output Interface Unit	32
	Bus Structure	33
	AM 2900 Chip Set	33
	AM 2901A	33
	AM 2902	34
	AM 2910	34
	AM 2914	35
	Summary	35
IV.	Arithmetic and Logic Unit Design	36
	ALU Computation Unit	36
	Two Port Ram	36
	Arithmetic Element	41
	Support Hardware	43
	Shift Multiplexers	43
	Carry-In Multiplexer	45
	ALU Flip Flops	46
	EX Flip Flop	49
	External Registers	50
	Shift Register Pairs	50
	Exponent Storage	50
	Program Control	51
	Summary	51
V.	Computer Control Unit Design	52
	Microprogram Sequencing	52
	Control Memory	52

Table of Contents

Page

Pipeline Register 54
Microsequencer. 56
Condition Test Multiplexer. 60
Microsequencer Operations 60
N-Counter 65

Instruction Decoding 66
J-K Multiplexers. 67
AB Latches. 67
Bit Mask Decoder. 67

Status Register. 68

Summary. 70

VI. Interrupt Control Unit Design. 71

AM 2914 Block Diagram Description. 71

Interrupt Control Unit Design. 74

ICU Microinstructions. 77
Microinstruction Descriptions 77
Microinstruction Comments 79

Summary. 80

VII. Input/Output Interface Unit Design. 81

I-BUS. 81
I-BUS Transfer Function 81
DMA Control Function. 84

Discrete I/O Channel Interface 89

Interval Timers. 91

Processor Control Panel. 93

Summary. 95

VIII. Microprogramming. 96

Microprogram Control Flow. 96
Power Up. 96
Instruction Fetch 96
Execution 100
Interrupt Handling. 102

Address Mode Decoding. 102

Summary. 112

Table of Contents

Page

IX. Evaluation 113
- Processor Speed 113
  - Instruction Execution Times. 113
  - Operational Speed. 113
  - Discussion 122
- Parts Count 122
- Summary 124

X. Conclusions and Recommendations. 125
- Conclusions 125
- Recommendations 126
  - Design Improvements. 126
  - Interrupt Changes. 126
- Comments. 127

Bibliography 128

Appendix A 129

Appendix B 156

Appendix C 168

Vita 219

List of Figures

Figure		Page
1	Representation of Single Precision Integer Data.	5
2	Representation of Double Precision Integer Data.	6
3	Representation of Floating Point Data.	7
4	Status Word Format	9
5	Basic Interrupt System Linkage	17
6	Microcomputer Block Diagram.	20
7	Processor Block Diagram.	31
8	Arithmetic and Logic Unit Block Diagram.	37
9	ALU Computation Unit Interconnection Diagram	38
10	ALU Computation Unit Functional Representation	39
11	ALU Shift Linkage Multiplexers	44
12	ALU Flip Flops Interconnection Diagram	47
13	Computer Control Unit Block Diagram.	53
14	Microinstruction Word Format	55
15	AM 2910 Microsequencer Block Diagram	57
16	Condition Test Multiplexer Block Diagram	61
17	Status Register and Condition Status Block Diagram . . .	69
18	AM 2914 Block Diagram.	72
19	Interrupt Control Unit Block Diagram	75
20	Input/Output Interface Unit Block Diagram.	82
21	I-BUS Interface for IOIU	83
22	Sequence Controller for I-BUS Interface.	87
23	DMA Controller for I-BUS Interface	88

List of Figures

Figure		Page
24	Discrete Data Line Interface	90
25	Interval Timers Block Diagram	92
26	Control Panel Block Diagram	94
27	Microprogram Control Flow	97
28	Power UP Microroutine	98
29	Instruction Fetch Microroutine	99
30	Execution Microroutine Example Double Precision Integer ADD	101
31	Interrupt Handling Microroutine	103
32	Address Mode Subroutines	107
33	Typical Decoding of R Address Mode	110
34	Typical Decoding of B Address Mode	111
B-1	Microinstruction Word Format	157

List of Tables

Table		Page
I	Instruction Word Format	10
II	Address Mode Description Derived Operand and Derived Address	11
III	Interrupt Vector Table.	15
IV	I-BUS Signal List	22
V	Baseline Avionic Instruction Mix.	28
VI	I/O Mapping Prom.	59
VII	I-BUS Interface Control Sequence.	85-86
VIII	Instruction Execution Times	114-120
IX	Baseline Avionic Instruction Mix.	121
X	Estimated Parts Count	123
A1	Op Code Matrix.	131
A2	Instruction Word Format	132
A3	Address Mode Description Derived Operand and Derived Address	133
A4	Instruction Set Description	134-155
B1	ALU Source Field.	159
B2	ALU Function Field.	159
B3	ALU Destinction Field	160
B4	Carry-In Field.	160
B5	Shift Multiplexer Field	161
B6	External Shift Field.	161
B7	Register Out Field.	162
B8	Register In Field	162
B9	Command Field	163

List of Tables

Table		Page
B10	Input/Output Interface Unit Field	164
B11	JK Multiplexer Field.	164
B12	AB Latch Field.	165
B13	Next Address Field.	165
B14	Condition Test Field.	166
B15	Interrupt Control Field	167
C1	Emulation Microprogram.	169-218

Abstract

This report presents the design of a microprogrammable processor which emulates the baseline instruction set of the Software-Compatible Avionic Computer Family. The general architecture of this processor includes 16/32 bit word lengths, sixteen (16-bit) general purpose registers, hardware fixed point arithmetic, firmware floating point arithmetic, and hardware vectored interrupts. The processor uses a flexible internal bus (I-BUS) to interface with memory modules, programmed input/output channels, and DMA channels. Integrated circuit devices from the AM 2900 family are used to realize the design. The processor's arithmetic and logic unit contains four AM 2901A bit slice devices which are cascaded by an AM 2902 carry look ahead generator. The processor's control unit uses an AM 2910 microsequencer to address control memory in a first level pipeline mode. The processor's hardware vectored interrupt system is managed by two AM 2914 priority interrupt encoders. Evaluation of the design determined that the processor's operational speed is 172 KOPS based upon a baseline avionic instruction mix. This operational speed falls below the design goal of 200 to 500 KOPS. The report concludes with recommendations for improving the processor's operational speed by adding hardware to facilitate floating point arithmetic and to pipeline I/O transfers over the I-BUS.

DESIGN OF THE PROCESSOR FOR
SOFTWARE COMPATIBLE AVIONICS COMPUTER FAMILY

I. Introduction

Background

The Air Force Avionic Laboratory (AFAL) has initiated a project to define specifications for a software-compatible avionics computer family. The purpose of this project is to provide a technical foundation for development of a family of future avionic processors which vary in size and performance, but share a common architecture and recognize a common baseline instruction set. Development of this computer family is expected to reduce future avionic development costs by reducing hardware design costs, by allowing use of common support software, and by providing opportunities to share mission software.

The general architecture proposed for the family has 16/32 bit word lengths, sixteen (16-bit) general registers, two's complement arithmetic, and hardware vectored interrupts. In addition, the family will accommondate programmed I/O channels; direct memory access channels; and a directly addressable memory state of 64K, 16-bit words. It is intended that the architecture be expandable for future growth of the family while retaining compatibility with earlier models.

The software-compatible instruction set is currently being developed with two processors in mind: a small processor for dedicated applications which implements a baseline instruction set, and a medium sized processor for centralized, higher throughput applications which implements the full

instruction set. The baseline instruction set for the smaller processor has been defined. The full instruction set for the medium sized machine is still being developed.

A major consideration in developing the software compatible instruction set is to retain compatibility with existing software developed for the Digital Avionic Information System (DAIS). In this regard, the proposed baseline instruction set represents an expansion of the existing DAIS (AN/AYK-15) instruction set.

The AFAL is currently evaluating the proposed baseline instruction set to identify problems in hardware implementation. The next step in that evaluation is to design a processor which interprets the proposed instruction set.

Statement of the Problem

The purpose of this thesis is to design a microprogrammable processor, using current state-of-the-art LSI integrated circuit devices, which emulates the baseline instruction set proposed for the software-compatible avionic computer family. This processor must also comply with the architectural requirements defined by the Air Force Avionics Laboratory (Ref.1).

Objective and Scope

The principal objective of this design is to assist the AFAL in evaluating the proposed baseline instruction set. This design will provide a basis for: evaluating the difficulty in interpreting each instruction; estimating the execution speed of each instruction; and estimating the hardware costs (i.e. device count). In addition, this design will determine if significant hardware savings can be realized by

deleting any group or class of instructions from the baseline specifications.

The scope of this thesis is limited to the design of the computer's central processor. The design of DMA channels, programmed I/O channels, memory modules, and other microcomputer components will not be included in this design. Within that context, the specific tasks included in the processor design are outlined below:

1) Define the processor's internal architecture and functional subunits.
2) Develop a functional design for each of these subunits.
3) Define the microinstruction word format for the processor.
4) Develop a microprogram to emulate the proposed baseline instruction set.

General Approach

To minimize parts count and hardware interconnections the processor will be designed using bit slice devices from the AM 2900 integrated circuit family. The AM 2900 family was selected for this major role because of its flexible architecture, high speed Schottky TTL technology, second source availibility, and wide range of support devices.

Thesis Outline

The processor requirements defined by the Air Force Avionic Laboratory are summarized in Chapter II. These requirements are followed, in Chapter III, by a brief overview of the processor design which identifies the major subunits of the processor. Chapters IV, V, VI, and VII describe the design of these subunits. The processor's microprogramming is described in Chapter VIII. Chapter IX presents an evaluation of the processor's design. Conclusions and recommendations are presented in Chapter X.

These chapters are supported by three Appendices. Appendix A describes the proposed software-compatible instruction set. Appendix B describes the microinstruction word format and microinstruction fields. The processor's microprogramming is documented in Appendix C.

II. Processor Requirements

The micro-computer processor described in this report was designed according to the specifications defined by the Air Force Avionics Laboratory (AFAL) in reference 1. This chapter summarizes those specifications.

Data Formats

The basic machine word length is 16 bits. The individual bits of the machine word are numbered so that the most significant bit is referred to as Bit 0 and the least significant bit of the word is referred to as Bit 15. Three data formats used by the processor are single precision integer, double precision integer, and floating point.

Single Precision Integer. The basic data format is single precision integer which represents data as a 16-bit signed two's complement number. Figure 1 shows the representation for the single precision integer and gives some examples.

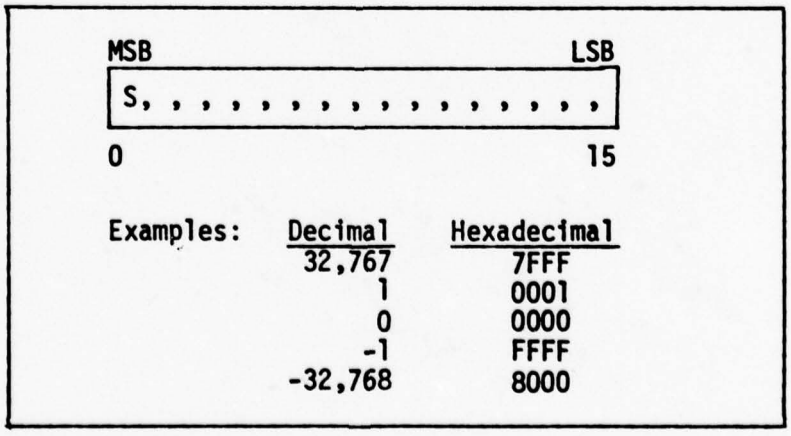
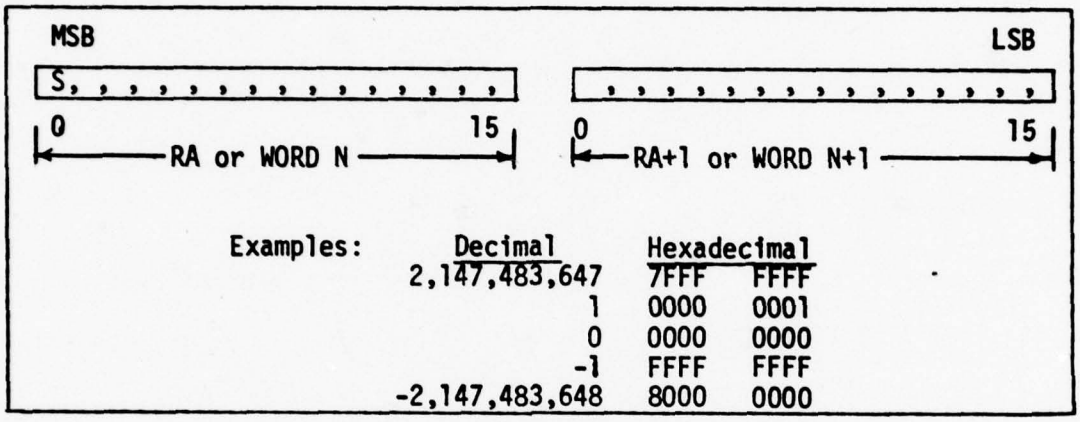


Fig. 1. Representation of Single Precision Integer Data

Double Precision Integer. The double precision integer format uses two sequential registers or memory words to present a 32-bit signed two's complement number. The most significant half of the 32-bit quantity is located in the lower numbered register or memory word. Figure 2 illustrates the format for double precision integers and provides some examples.

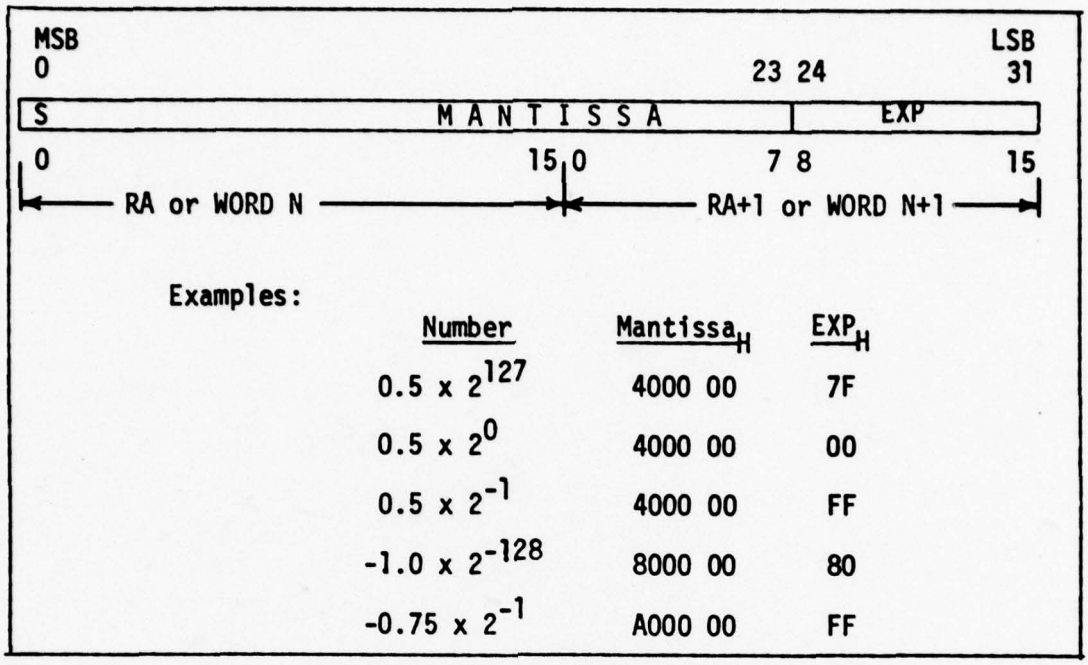


Examples:	Decimal	Hexadecimal	
	2,147,483,647	7FFF	FFFF
	1	0000	0001
	0	0000	0000
	-1	FFFF	FFFF
	-2,147,483,648	8000	0000

Fig. 2. Representation of Double Precision Integer Data

Floating Point. The floating point data format is a 32-bit quantity that uses two sequential registers or memory words. The characteristic (exponent) is an 8-bit, signed two's complement number which occupies the 8 least significant bits of the 32 bit word (bits 24 through 31). The mantissa is a 24-bit signed two's complement number and occupies the 24 most significant bits of the 32 bit word (bits 0 through 23). Figure 3 shows the representation for floating point numbers.

A normalized floating point number is defined to have a mantissa in which the sign bit (bit 0) and the next most significant bit (bit 1) are not equal. The normalized representation for the mantissa of negative powers of 2 (-1.0×2^n) is the smallest 24-bit two's complement number, $8000\ 00_H$. Floating point zero is represented as 0.0×2^{-128}, which is $000\ 0080_H$.



Examples:

Number	$Mantissa_H$	EXP_H
0.5×2^{127}	4000 00	7F
0.5×2^{0}	4000 00	00
0.5×2^{-1}	4000 00	FF
-1.0×2^{-128}	8000 00	80
-0.75×2^{-1}	A000 00	FF

Fig. 3. Representation of Floating Point Data

Internal Registers

General Register File. The processor contains a user addressable general register file comprised of sixteen 16-bit registers (R0,R1,..., R15). These general registers may be used as accumulators, index registers, base registers, temporary operand memory, and stack pointers with the following restrictions:

1. Only registers R1, R2,...,R15 may be used as index registers.

2. Only even numbered registers (R0,R2,...,R4) may be specified as accumulators for double precision or floating point instructions.

3. Only four registers (R4, R5, R6, R7) may be used as base registers for instructions using the Base Relative address mode.

4. Instructions which use the Base Relative address mode reference an implied accumulator. The register pair (R0, R1) is the implied accumulator for double precision and floating point operations. Register R2 is the implied accumulator for single precision integer operations.

5. Register R15 is the implicit stack pointer for the Push and Pop Multiple instructions.

Instruction Counter. The Instruction Counter (IC) is a 16-bit register that is used for program sequencing. It allows instructions within a range of 64K words to be executed. The Instruction Counter is internal to the processor but it is not part of the general register file.

Status Word. The processor contains a 16-bit status word (SW) register which defines the state of the processor. The first 4-bit field of the status word is dedicated to instruction results (i.e. Instruction Status Bits) and is referred to as the Instruction Status (IS). The second 4-bit field of the status word is dedicated to error conditions (i.e. Error Status Bits) and is referred to as Error Status (ES). The remaining 8-bit field of the status word is undefined and is reserved for future defined program status (PS) such as privileged mode status, interrupt status, etc. Figure 4 shows the format of the status word and defines the IS field and the ES field.

Interrupt Mask. The Interrupt Mask (IM) register contains the 16-bit Interrupt Mask that specifies which interrupts are currently enabled and which are disabled.

Working Registers. The processor also contains a number of working registers (EAR,N,W,X,Y,Z,G,H,Q, and IR) which are not directly accessable to software. The function and operation of these working registers will be discussed later, when the processor's architecture is described.

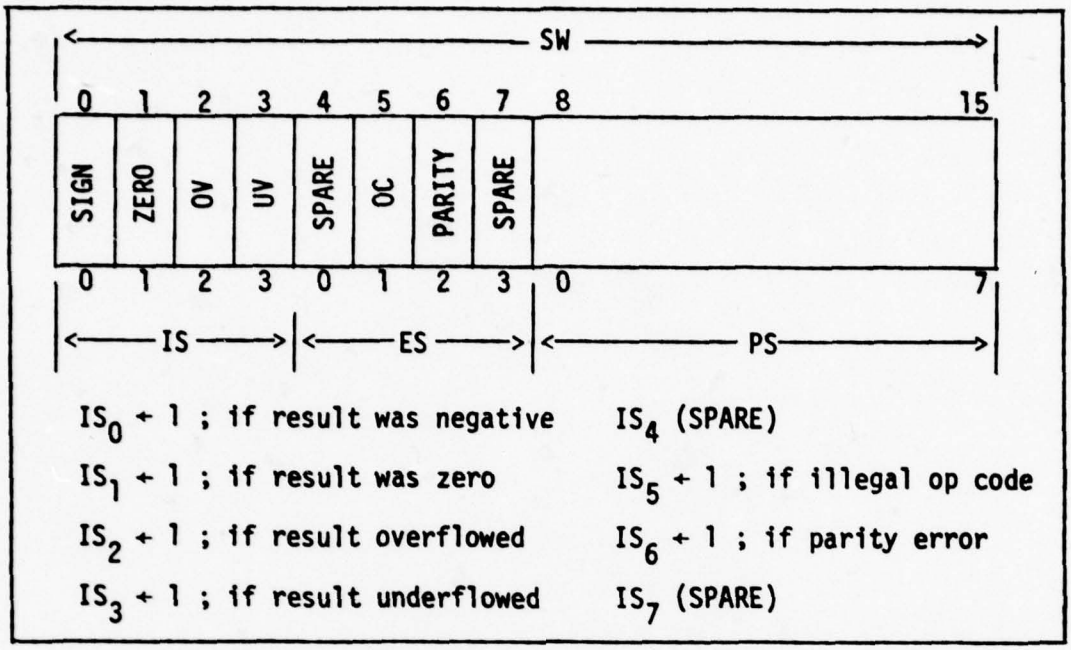


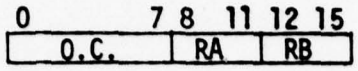
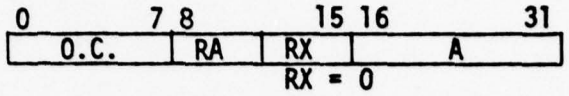
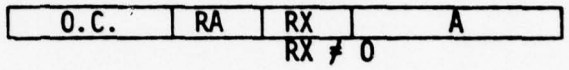
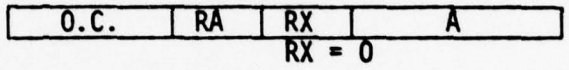
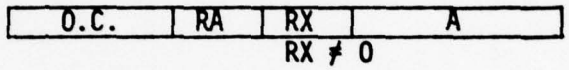
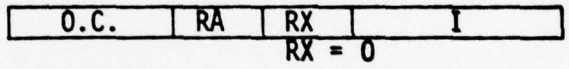
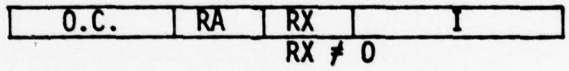
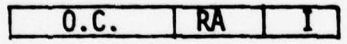
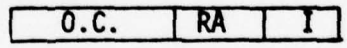
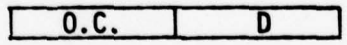
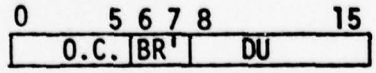
Fig. 4. Status Word Format

Address Modes

Table I describes the instruction word format for each address mode used by the processor. The Derived Address (DA) and Derived Operand (DO) associated with each of the addressing modes is shown on Table II. The following paragraphs discuss each of the addressing modes in more detail.

Register Direct. Instructions with the Register Direct (R) address mode use the general purpose register specified by the RA field as the accumulator. Field RB indicates which of the general purpose registers contains the required operand. When the R address mode is used with two word floating point or double precision instructions, fields RA and RB must refer to even numbered registers.

Table I
Instruction Word Format

ADDRESSING MODE	SYMBOL	FORMAT
1. Register Direct	R	
2. Memory Direct Non-Indexed	D	
3. Memory Direct Indexed	DX	
4. Memory Indirect Non-Indexed	I	
5. Memory Indirect Indexed	IX	
6. Immediate Long Non-Indexed	IM	
7. Immediate Long Indexed	IMX	
8. Immediate Short Positive	ISP	
9. Immediate Short Negative	ISN	
10. IC-Relative	ICR	
11. Base Relative	B	

BR = BR' + 4
Implied Accumulator:
F.P. = (R0, R1)
D.P. = (R0, R1)
S.P. = R2

Table II
Address Mode Description
Derived Operand and Derived Address

ADDRESS MODE	SYMBOL	DERIVED OPERAND (DO)		DERIVED ADDRESS (DA)	
		S.P.	Flt.P. & D.P.	S.P.	Flt.P. & D.P.
Register Direct	R	(RB)	(RB,RB+1)	RB	RB,RB+1
Memory Direct:					
Non-Indexed	D	{A}	{A,A+1}	A	A,A+1
Indexed	DX	{A+(RX)}	{A+(RX),A+1+(RX)}	A+(RX)	A+(RX),A+(RX)+1
Memory Indirect:					
Non-Indexed	I	{{A}}	{(A),(A)+1}	{A}	{A},{A}+1}
Indexed	IX	{{A+(RX)}}	{{A+(RX)},{A+(RX)+1}}	{A+R(X)}	{A+(RX)},{A+(RX)+1}
Immediate Long					
Non-Indexed	IM	I	-	-	-
Indexed	IMX	I+(RX)	-	-	-
Immediate Short					
Positive	ISP	+(I+1)	-	-	-
Negative	ISN	-(I+1)	-	-	-
IC-Relative	ICR	-	-	D+(IC)	-
Base Relative	B	{DU+(BR)}	{DU+(BR),DU+1+(BR)}	DU+(BR)	DU+(BR),DU+1+(BR)

Note: Abbreviations used on this table are defined in Appendix B.

Memory Direct. In the Memory Direct (D) address mode, an instruction specified memory address (A) contains the required operand. Field RA denotes which general purpose register will be used as the accumulator. The RX field will always be zero for this addressing mode.

Memory Direct-Indexed. The Memory Direct-Indexed (DX) address mode is the same as the D address mode except that the required operand is specified by the sum of the instruction address field (A) and the contents of the index register (RX). Any of the general purpose registers except R0 can be specified as the index register by the RX field.

Memory Indirect. In the Memory Indirect (I) address mode, an instruction specified memory address (A) contains the address of the required operand. Field RA denotes which of the general purpose registers will be used as the accumulator and the RX field must be zero.

Memory Indirect with Pre-Indexing. The sum of the contents of the specified index register (RX) and the instruction address field (A) is the address of the memory location which contains the address of the required operand. Any of the general purpose registers except R0 can be used as an index register by the RX field. Field RA identifies which register will be used as the accumulator.

Immediate Long. The immediate operand for the Immediate Long (IM) address mode is specified by the I field of the instruction. As in previous address modes, the accumulator is defined by the RA field and the RX field must be zero.

Immediate Long-Indexed. The required operand for the Immediate Long-Indexed (IMX) address mode is obtained by adding the contents of the index register specified by the RX field to the value of the immediate I field. The accumulator is identified by the RA field and the RX field must not be zero.

Immediate Short Positive. The immediate operand for the Immediate Short Positive (ISP) address mode is a positive integer between 1 and 16. This operand is encoded into the 4-bit I field of the instruction and is obtained by adding one to the value of the I field as shown in Table 1 . The accumulator is identified by the RA field.

Immediate Short Negative. The Immediate Short Negative (ISN) address mode is similar to the ISP address mode except that the immediate operand is a negative integer between -1 and -16. This negative operand is obtained from the instruction by adding one to the value of the I field and then interpreting the result as a negative integer.

IC-Relative. The IC-Relative (ICR) address mode is used for 16-bit branching instructions. The contents of the instruction counter (i.e., the address of the current instruction) is added to the sign extended 8-bit displacement field (D) of the instruction. The sum points to the memory address to which program control may be transferred if a branch is executed. This allows addressing within a memory region of -128_{10} to $+127_{10}$ words relative to the address pointed to by the instruction counter (IC).

Base Relative. The Base Relative (B) address mode is used to address a memory region of $+256_{10}$ words beginning at the address pointed to by the base register (BR). The general purpose register (R4, R5, R6, or R7) which is used as the base register is identified by adding 4_{10} to the value of the 2-bit BR' field of the instruction. The 8-bit displacement (DU) field of the instruction is interpreted as a positive integer between 0_{10} and 256_{10}. The memory address of the required operand is obtained by adding the value of the displacement field to the contents of the base register. The implied accumulators for the floating point and double precision operations are registers R0 and R1. The implied accumulator for

single precision operations is register R2.

Instruction Set

As stated in the introduction, the processor implements the instruction set which has been proposed for the software-compatible avionic computer family. This instruction set is basically an extension of the current AN/AYK-15 Digital Avionic Information System (DAIS) instruction set. It is software compatible with existing DAIS software, yet it provides numerous features not available in the DAIS instruction set. A full description of the processor instruction set is too lengthy for inclusion in this chapter and is therefore presented in Appendix A.

Interrupt System

The processor's interrupt system is hardware vectored with sixteen levels of priority. Interrupt 16 has the highest priority and Interrupt 1 has the lowest priority. The even numbered interrupts are dedicated to internally generated interrupts and the odd numbered interrupts are dedicated to externally generated interrupts. Table III shows how these sixteen interrupt levels are assigned.

Masked or disabled interrupt requests are not ignored. Interrupt service hardware includes a 16-bit Interrupt Pending Register which captures interrupts regardless of whether they are masked or disabled. Once an interrupt is captured, it remains latched in the Interrupt Pending Register until it is recognized or the register is cleared.

Interrupt requests may occur at any time, but interrupt processing must wait until the current instruction is complete. Once an interrupt is recognized, other interrupts (except Interrupt 16) are automatically

Table III
Interrupt Vector Table

		Interrupt	Linkage Pointer $_H$	Service Pointer $_H$
Lowest Priority	1.	Spare (External)	20	21
	2.	Underflow/Overflow	22	23
	3.	Spare (External)	24	25
	4.	Illegal Operation Code	26	27
	5.	Spare (External)	28	29
	6.	Spare (Internal)	2A	2B
	7.	Spare (External)	2C	2D
	8.	Interval Timer (100 us)	2E	2F
	9.	Spare (External)	30	31
	10.	Interval Timer (10 us)	32	33
	11.	Spare (External)	34	35
	12.	Processor Parity Error	36	37
	13.	Spare (External)	38	39
	14.	Spare (Internal)	3A	3B
	15.	Spare (External)	3C	3D
Highest Priority	16.	Power Down	3E	3F

disabled until the Enable Interrupt instruction (described below) is executed. Interrupt 16 is reserved for power down and cannot be masked or disabled except for testing purposes. Interrupts 14 and 15 cannot be disabled by the Disable Interrupts instruction (described below); but they may be selectively masked.

The 16 bit Mask Register is used to selectively disable individual interrupt levels. The bits in the mask register are assigned so that Interrupts 1 through 15 are masked by bits 15 through 1 of the mask register, respectively. A "one" in a mask bit disables the corresponding interrupt level and a "zero" in a mask bit enables the corresponding interrupt level. For example, a "one" in mask bit 15 will disable recognition of interrupt requests for Interrupt 1.

For each interrupt level there are two fixed memory locations in the "vector table". The first memory location ("Linkage Pointer") points to the memory address that may be used (option) to store the current (old) state of the computer (i.e., IM, SW, and IC). The second memory location ("Service Pointer") points to the memory address that contains the next (new) state of the computer.

Saving Old Computer State. The processor allows the user the option of storing the old computer state (current IM, SW, and IC) in either a memory stack or the memory address pointed to by the Linkage Pointer. When an interrupt is recognized, the processor hardware tests the Linkage Pointer for a "zero" value. If the Linkage Pointer is not "zero", the old computer state is saved in memory starting with the memory address identified by the Linkage Pointer as shown in Figure 5. If the value in the Linkage Pointer is equal to "zero", general purpose register R15

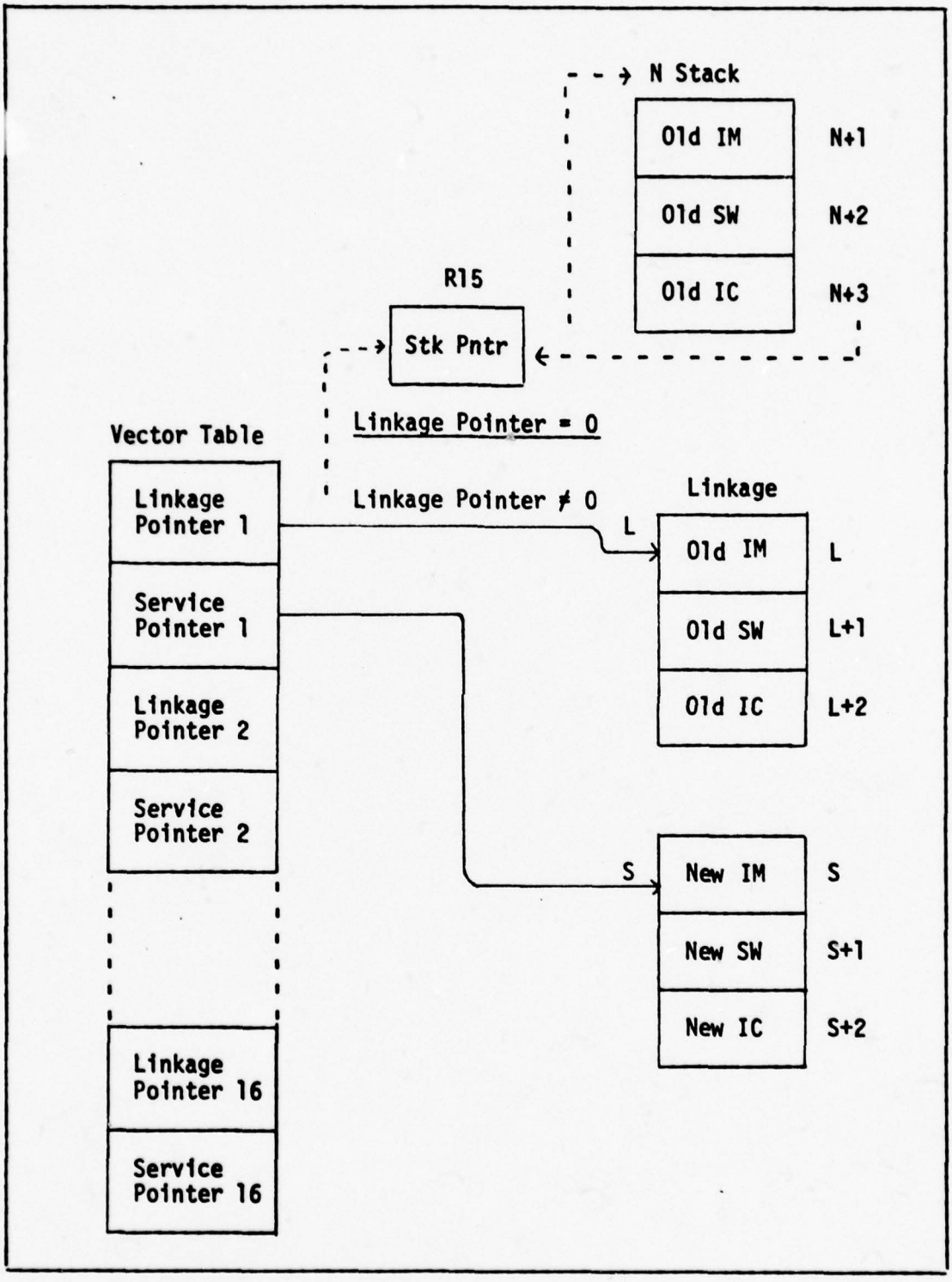


Fig. 5. Basic Interrupt System Linkage

is used as a stack pointer for saving the old processor state (IM, SW, and IC) on memory stack as shown below:

$$\{(R15) + 1\} \leftarrow IM$$

$$\{(R15) + 2\} \leftarrow SW$$

$$\{(R15) + 3\} \leftarrow IC$$

$$(R15) \leftarrow (R15) + 3$$

New Computer State. Regardless of the option used to save the old computer state, the new computer state is always fetched from main memory starting at the memory address identified by the Service Pointer (see Figure 5).

Return from Interrupt. There are two ways to accomplish a return from interrupt. If the old computer state was saved in memory starting at the address identified by the Linkage Pointer, then return from interrupt is accomplished by executing a Load Status (LDST) instruction with the address value from the Linkage Pointer. However, if the old computer state was saved on a memory stack, then return from interrupt must be accomplished by executing a Return from Interrupt (RFI) instruction.

Interrupt Input/Output Instructions. The interrupt unit responds directly to the following Input/Output instructions:

(1) Interrupt Disable. When the Interrupt Disable (DSBL) instruction is executed, recognition of all interrupt requests (except Interrupt 14, Interrupt 15, and Interrupt 16) is disabled.

(2) Interrupt Enable. When the Enable Interrupt (ENBL) instruction is executed, the interrupt system will continue to disable interrupts until the end of the next instruction. Then non-masked interrupts will be recognized.

(3) Clear Interrupts. The Clear Interrupt (CLIR) instruction clears all pending interrupts from the Interrupt Pending Register.

(4) Set Interrupt Control. The Interrupt Mask Register is read when the Read Interrupt Control (RIC) instruction is executed.

Microcomputer I-BUS

The microcomputer is organized around a single, bi-directional system bus which interconnects the processor and other microcomputer subsystems such as Memory Units (MU) and Programmed Input/Output (PIO) channels. This internal bus structure (I-BUS), shown in Figure 6, is similar to the I-BUS defined for the Distributed Processor/Memory (DP/M) system (Ref 2:156-166) except for the control signals associated with the interrupt system operation. In the DP/M system, the interrupt structure is distributed between various DP/M subsystems and interrupt processes such as interrupt request, acknowledge, and external interrupt vector transmission, are coordinated via I-BUS control signals. The interrupt system used by this microcomputer is hardware vectored and operates independently from the I-BUS.

Devices interfaced to the I-BUS compete for access on a priority basis. High-speed peripherals are usually assigned highest priority and the processor is assigned the lowest. The I-BUS is asynchronous and the speed of data transfers over it are determined by the speed of the devices to which it is interfaced.

There are two classes of devices which interface to the I-BUS: master devices, which control data transfers; and slave devices, which generate or accept data in response to some master. Data transfers in either direction always occur between one master and one slave. The processor is an

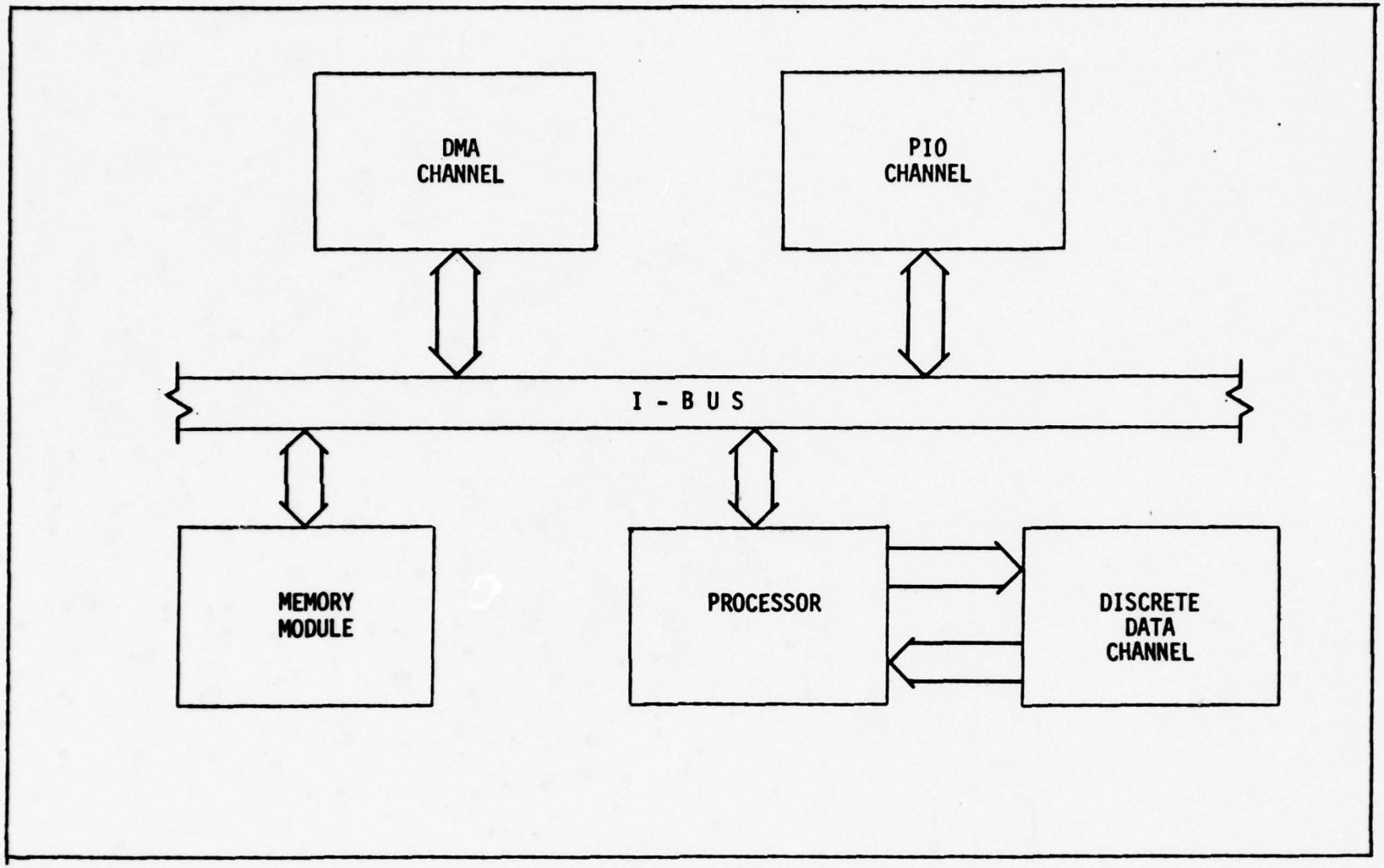


Fig. 6. Microcomputer Block Diagram

example of a master device and a memory module is an example of a slave device. All slave devices recognize and are activated by specific addresses. For example, a memory module is activated when some master device does a memory data receive to an address within the boundaries of that memory module.

Table IV shows the various signals associated with the I-BUS. As indicated there are four categories of I-BUS lines: Bus Master Control; Data Transfer and Control; General Bus Facilities; and Special Functions. Each of the categories are defined in the following paragraphs.

Bus Master Control. The three bus master control signals are bus request (BRQ), bus release (BREL), and bus grant (BGR). The bus request line (BRQ) is used to initiate a bus master assignment, while the bus release line (BREL) is used to terminate the assignment. The bus grant line (BGR) is threaded through each module and is used to resolve conflicts when two or more masters request the bus simultaneously. A master device can request the I-BUS only when bus grant (BGRI) is FALSE. When the bus request line (BRQ) is asserted, the bus grant (BGR) 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 (BGRO) signal from propagating to any lower priority module which may also be requesting the bus. When the master initiates a transfer (issues the TRQ signal), it also issues a bus release (BREL). This command forces all bus masters, including itself, to remove their bus requests (BRQ). When the master sees BGRI go FALSE, it releases BREL and unblocks BGRO. Block transfers are accomplished by immediate re-requests of the bus. A device doing a block transfer can always be preempted between transfers by a higher priority device

Table IV
I-BUS Signal List

Signal Type	Signal Name	Number of Wires
Data Transfer and Control	DATA (DATA)	16
	ADDRESS (ADDR)	16
	I/O Select (IOSL)	1
	DATA Receive (DRCV)	1
	Transfer Request (TRQ)	1
	Transfer Time Out (TTO)	1
	Transfer Acknowledge (TACK)	1
Bus Master Control	Bus Request (BRQ)	1
	Bus Release (BREL)	1
	Bus Grant In (BGRI)	1
	Bus Grant Out (BGRO)	1
	Master ID (BMID)	4
General Bus Facilities	Free Running Clock (FCLK)	1
	Master Clock (MCLK)	1
	Master Stop (MSTP)	1
	Clear/Reset (CLR)	1
	Power/Ground	as required
Special Functions	Processor Status (PSTAT)	1
	DMA Disable (DMAD)	1

NOTE: For all I-BUS signals:
"Low" is "TRUE"
"High" is "FALSE"

requesting the bus when BREL goes FALSE. A new bus master will wait until the bus is quiescent (transfer request (TRQ) and transfer acknowledge (TACK) have ended) and then start its transfer cycle. This sequence allows bus mastership resolution to be overlapped with data transfers, thus minimizing or partially eliminating the overhead associated with bus master assignments. The four identification lines are used to identify which master is in control of the bus (Processor ID = 0; the default value).

Data Transfer and Control. Data transfer and control lines are used exclusively for data transfer operations on the I-BUS. These control lines can be best described by reviewing the two types of data transfers: Master to Slave send cycle; and Master from Slave receive cycle.

Master to Slave Send Cycle. When an I-BUS master device has access to the I-BUS, it accomplishes a send cycle through the following actions. The master, after gaining I-BUS control, asserts transfer request (TRQ). At the same time the master also asserts a send command via data receive (DRVC) and specifies whether it is a memory or I/O operation (IOSL). Concurrent with these actions the master supplies 16-bits of valid data (DATA) and a valid 16-bit address (ADDR). All slave devices interfaced to the I-BUS will receive the transfer request (TRQ) transmitted by the master and will decode the address (ADDR) to determine which slave is being addressed. When the slave device which is being addressed has decoded the address as valid, it will assert a transfer acknowledge (TACK). At the same time the slave device asserts transfer acknowledge, it will either clock the data (DATA), address (ADDR), data receive (DRVC), and I/O select (IOSL) signals from the I-BUS into registers, or in the case of memory, delay the transfer acknowledge (TACK) until the memory write cycle is

complete. When the I-BUS master device receives the asserted transfer acknowledge (TACK), it releases the transfer request (TRQ). The master then looks for the release of the transfer request (TRQ) line before releasing data receive (DRCV), I/O select (IOSL), address (ADDR), and data (DATA). This allows a slower third party device, such as a maintenance panel, to delay the transfer while it latches the data and/or address for monitor purposes. When the slave device receives the release of the transfer request (TRQ), it releases the transfer acknowledge (TACK). When the master device receives the release of the transfer acknowledge (TACK), it may begin a new cycle or it may relinquish the I-BUS to another master device.

Master from Slave Receive Cycle. When an I-BUS master device has access to the I-BUS, it will accomplish a receive cycle through the following action. The master asserts transfer request (TRQ), and supplies a valid 16-bit address (ADDR). At the same time the master specifies whether it is a memory or I/O operation (IOSL). All slave devices interfaced to the I-BUS receive the transfer request and decode the address (ADDR). When the slave device which is being addressed decodes the address as valid it will begin to send data. When the data (DATA) is valid, the slave device asserts transfer acknowledge (TACK). When the I-BUS master device receives the TACK, it internally delays it to account for the worst case I-BUS skew and then releases transfer request (TRQ). As the master releases TRQ, it clocks the data (DATA) from the I-BUS into its internal register. The master then looks for the release of the transfer request (TRQ) line before releasing data receive (DRCV), I/O select (IOSL), and address (ADDR). This allows a slower third party device to delay the transfer as described previously. When the slave device receives the released transfer request (TRQ), it then releases transfer

acknowledge (TACK) and data (DATA). When the master receives the released transfer acknowledge (TACK) the transfer is complete.

In addition to the normal transfer request/acknowledge sequence there is one way in which a transfer can be terminated abnormally. A transfer time out (TTO) is generated by a watchdog timer. 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 error flag and proceeds.

General Bus Facilities. The general bus facilities are comprised of a free running clock (FCLK), master clock (MCLK), a master stop (MSTP), a clear/reset (CLR) signal, power and ground. The free running clock is intended for slave devices with dynamic memories which require a continuous clock to perform their refresh operation. The master clock (MCLK) is used by the processor and other master devices. The master clock and free running clock are similar except that the master stop (MSTP) signal will inhibit the master clock without affecting the free running clock. The clear-reset (CLR) signal is used to initialize the processor and all other devices. The bus becomes quiescent when the clear/reset (CLR) is used. When the clear-reset is issued, all activity on the bus aborts, and all signals, except BREL, go FALSE: BREL stays TRUE until CLR becomes FALSE.

Special Functions. These are two special function signals associated with the processor: Processor Status (PSTAT); and DMA Disable (DMAD). Processor status is used in conjection with the data transfer signals to indicate the state that the processor is in and the nature of the data (DATA) and address (ADDR). The processor status (PSTAT) is only TRUE when the processor is making a transfer. The DMA disable signal is used to

interleave the I-BUS operations of the processor and the DMA channel. The DMA disable (DMAD) signal is issued by the processor when the processor is idle waiting for access to the I-BUS and the DMA channel is the I-BUS master. This signal inhibits the DMA channel from generating a bus request (BRQ). When this signal is issued, the DMA channel will complete its current bus transfer and then relinquish control of the I-BUS. The processor will release this signal when it has gained control of the I-BUS. Since the DMA channel has higher priority than the processor, control of the I-BUS will return to the DMA channel once the processor completes a transfer.

Input/Output

The input/output sections of the microcomputer include a DMA Channel, a Programmed Input/Output Channel, and a Discrete Input/Output Channel (see Figure 6). The detailed design of these input/output channels is not included within the scope of this report. However, the requirements for interface and control signals between the processor and input/output channels are included in the processor design.

DMA Channel. The Direct Memory Access (DMA) Channel is implemented and controlled by signals external to the computer. The DMA operates on a "cycle steal" basis with the processor. The DMA channel is prevented from stopping the processor's operation by the requirement that the DMA yield to at least one processor memory access between each DMA access. Likewise, during multiple memory access instructions such as store multiple, the processor will minimize DMA delays by allowing the DMA channel to interleave its use of memory.

Programmed Input/Output Channel. The Programmed Input/Output (PIO) channel is a 16-bit bi-directional bus with associated control lines over

which processor software controlled input/output occurs. The PIO channel interfaces to the processor as an I-BUS slave device. It responds to the I-BUS control signals (TRQ, IOSL, and DRCV) and is activated by a specific address (ADDR).

Discrete Input/Output Channel. The microcomputer input/output section includes sixteen discrete lines (8 input and 8 output). Data transfers over these lines is controlled by I/O software instructions through data buffers.

Interval Timers

Two interval timers are included in the processor design. These timers are basically 16-bit counters which operate as follows: a one is added to the least significant bit of timer A every 10 micro-seconds and timer B every 100 micro-seconds. Both timers can be loaded (by output instruction) and read (by input instruction). An interrupt is generated when the timer increments from $FFFF_{Hex}$ to 0000_{Hex}. If these timers are not loaded, an interrupt is generated after $65{,}536_{10}$ counts.

Operational Speed Requirement

The required operational speed of the microcomputer is in the range of 200-500 KOPS based upon a Gibson mix of instructions. The instruction mix shown in Table V is considered to be representative of an avionic mix and shall be used as a basis for determining the final throughput of the microcomputer system.

Table V
Baseline Avionic Instruction Mix

INSTRUCTION TYPE	PERCENT
Floating Point:	
Add/Subtract	14
Multiply	5
Divide	1
Load/Store	10
Fixed Point:	
Add/Subtract:	
Memory Direct	1
Immediate	2
Memory Direct Indexed	1
Load:	
Memory Direct	22
Memory Direct Indexed	3
Register Direct	4
Store	9
Add Register Direct	1
Decrement and Branch	9
Conditional Branch	18
	100

Summary

This chapter described the requirements for the processor. It included a description of the processor's data representations, general purpose register, addressing modes, instruction set, interrupt handling, and input/output. The next five chapters describe the functional design of a microprogrammable processor which fulfills the processor requirements outlined in this chapter.

III. Design Overview

This chapter presents a brief overview of the processor's architecture as a prelude to subsequent chapters which describe the individual subunits of the processor in detail. This chapter will introduce the processor's subunits, describe their basic functions, and summarize the basic bus structure which transfers information between the subunits.

The basic architecture of the processor is shown on Figure 7. The processor has four functional subunits: the Arithmetic and Logic Unit (ALU), the Computer Control Unit (CCU), the Interrupt Control Unit (ICU), and the Input/Output Interface Unit (IOIU). These subunits are interconnected by three data busses: D-BUS, A-BUS, and Control BUS. Each of these components are described in the following paragraphs.

Arithmetic and Logic Unit

All arithmetic, logical, and shifting operations for the processor are performed in the Arithmetic and Logic Unit. The ALU is also responsible for performing program control functions such as effective address calculations and keeping track of the current instruction address in main memory. In addition to the basic computational hardware, the ALU also contains the sixteen general purpose registers and a variety of working (scratch pad) registers. A detailed description of the ALU is presented in Chapter IV.

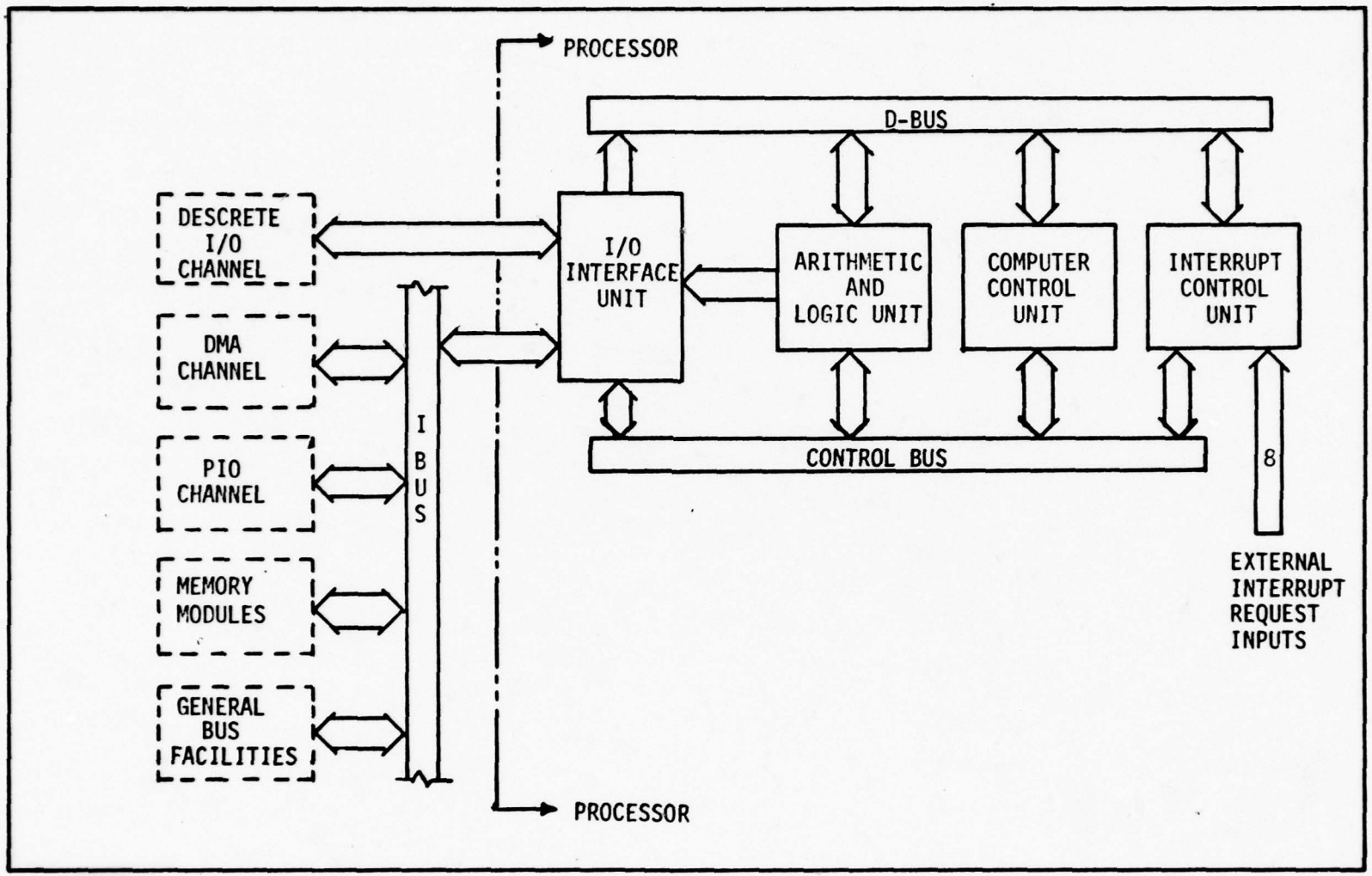


Fig. 7. Processor Block Diagram

Computer Control Unit

The Computer Control Unit (CCU) receives machine instruction words from memory, decodes them, and issues the appropriate control signals to the other sections of the processor to cause the desired operation to take place. The CCU is the most complex subunit of the processor. It contains the processor's instruction register, status register, two OP Code mapping PROMs, a microsequencer, microprogram control memory, a microloop counter, a pipeline microinstruction register, microinstruction decoders, and a variety of multiplexers. The Computer Control Unit is described in Chapter V.

Interrupt Control Unit.

The Interrupt Control Unit (ICU) manages the processor's hardware vectored interrupt system. It accepts and assigns priorities to 16 levels of interrupts. The ICU maintains the interrupt mask register, keeps track of pending interrupts, and generates the interrupt vector when an interrupt is recognized. The Interrupt Control Unit is described in Chapter VI.

Input/Output Interface Unit.

Data transfers between the processor and the other components of the microcomputer (i.e. memory modules, DMA Channel, and PIO Channels) are accomplished via the I-BUS described in Chapter II. The main function of the Input/Output Interface Unit (IOIU) is to interface the processor with the microcomputer's I-BUS. The IOIU also interfaces the processor with the Descrete Input/Output (DIO) Channel. In addition, the IOIU contains the interval timers (Timer A and Timer B) and the processor control panel. Chapter VII describes the IOIU.

Bus Structure

The four functional subunits of the processor communicate over three internal busses as shown on Figure 7.

(1) The Date Bus (D-BUS) is a 16-bit wide bidirectional three state bus which transmits data between the subunits of the processor.

(2) The Address Bus (A-BUS) is a 16-bit wide data path from the ALU to the address buffer of the Input/Output Interface Unit (IOIU).

(3) The Control Bus is comprised of all control signals which extend from the Computer Control Unit (CCU) to the other processor subunits and the status signals which the other subunits return to the CCU.

AM2900 Chip Set

As stated in the introductory chapter, bipolar LSI bit slice devices from the AM 2900 device family will be used as the basic building blocks for the processor design. Four AM 2901 four bit microprocessor slices and an AM 2902 look-ahead carry device will form the nucleus of the Arithmetic and Logic Unit. A single AM 2910 microcontroller chip is used as the microsequencer in the Computer Control Unit. The Interrupt Control Unit is designed around two AM 2914 Priority Interrupt Encoders.

Since the processor design, which is presented in the next four chapters, will assume a degree of familiarity with the AM 2900 device family, a brief description of these key components is presented below (Ref. 3).

AM 2901A. The AM 2901A is a 4-bit bipolar microprocessor slice which is designed for use as a CPU, peripheral controller, or numerous other applications. It incorporates a sixteen word 4-bit two-port Random Access Memory (RAM) with latches on both output ports, a high performance Arithmetic and Logic Unit (ALU) and shifter, a multipurpose Q-register with shift

inputs, and a nine bit microinstruction decoder. Although the AM 2901A is only 4-bits wide, it was designed as a cascadable element with provisions for either ripple carry or high speed look-ahead carry. Any number of AM 2901A elements may be cascaded together to form words of virtually any length. In this design, four AM 2901A devices will be cascaded to form a 16-bit microprocessor which will be referred to as the ALU Computational Unit (CU).

AM 2902. The AM 2902 is a high speed, look ahead carry generator which accepts up to four pairs of carry propagate and carry generate signals and provides the anticipated carries. One AM 2902 device will be used to generate the carry look-ahead signals for the four AM 2901A devices of the ALU Computation Unit (CU).

AM 2910. The AM 2910 Microprogram Controller is a 12-bit wide address sequencer intended for controlling the sequence of execution of microinstructions stored in microprogram memory. The key element of the AM 2910 is a multiplexer which outputs a 12-bit microinstruction address to microprogram memory from one of four sources: (1) the internal microprogram address register, which usually contains the previous microaddress incremented by one: (2) an external (direct) input: (3) an internal register "R" which contains data loaded during a previous micro-instruction: or (4) a five deep last-in, first-out internal stack. (Ref. 4).

Besides the capability of sequential access, the AM 2910 provides conditional branching to any microinstruction within its 4096-microword range. The last in, the first-out stack provides microsubroutine return linkage and looping capability for five levels of nested microroutines. The AM 2910 is controlled by a four bit microinstruction and executes sixteen different microsequence control instructions, most of which are conditional.

AM 2914. The AM 2914 is a high speed, eight-bit priority interrupt unit that is casadable to handle any number of priority interrupt request levels. Interrupt requests are received as either pulses or levels on eight interrupt input lines and these inputs are stored in an internal interrupt register. The circuit provides a built in mask register to mask individual interrupts and a built in status register which points to the lowest priority of interrupt that will be accepted. When enabled, the AM 2914 outputs a three bit binary coded vector which identifies the highest prioity, non-masked interrupt request received. The 16 level priority interrupt system for the processor described in this report will use two AM 2914 elements cascaded together.

Summary

This chapter was intended to assist the reader by providing the brief overview of the processor design. It described the four subunits of the processor and the bus structure which interconnects those units. In addition, this chapter introduced the AM 2900 chip set which will be used to implement the processor's design. The next four chapters will describe the processor's subunits in detail.

IV. Arithmetic and Logic Unit Design

The functional design of the processor's Arithmetic and Logic Unit (ALU) is presented in this chapter. The primary emphasis for this discussion is the ALU architecture and its response to specific micro-control signals.

The key element of the ALU's architecture, shown on Figure 8, is the Computation Unit (CU). The remaining components can be classified as either support hardware or as external registers. Because of the ALU's heavy dependence upon the Computation Unit (CU), the CU will be described first, its supporting hardware second, and the external registers last.

ALU Computation Unit

The heart of the ALU is a 16-bit wide Computation Unit (CU) which is built from four AM 2901A bit slice chips and an AM 2902 look-ahead carry device as shown on Figure 9. A functional representation of the CU's internal architecture is presented on Figure 10. This functional representation shows that the main components of the Computation Unit are a sixteen word 2-port Random Access Memory (RAM) and a high speed arithmetic and logic unit. To avoid confusion between reference to the processor's ALU and the Computation Unit's arithmetic and logic unit, the latter will be referred to as the Arithmetic Element (AE) of the Computation Unit.

Two Port RAM. The two port RAM contains sixteen 16-bit words which are used as the general purpose register file (R0, R1, ... , R15). Data in any of the sixteen words of RAM can be read from the A-port of the RAM by applying a 4-bit binary address to the A-address input. Likewise, any of the sixteen words can be simultaneously read from the B-port by

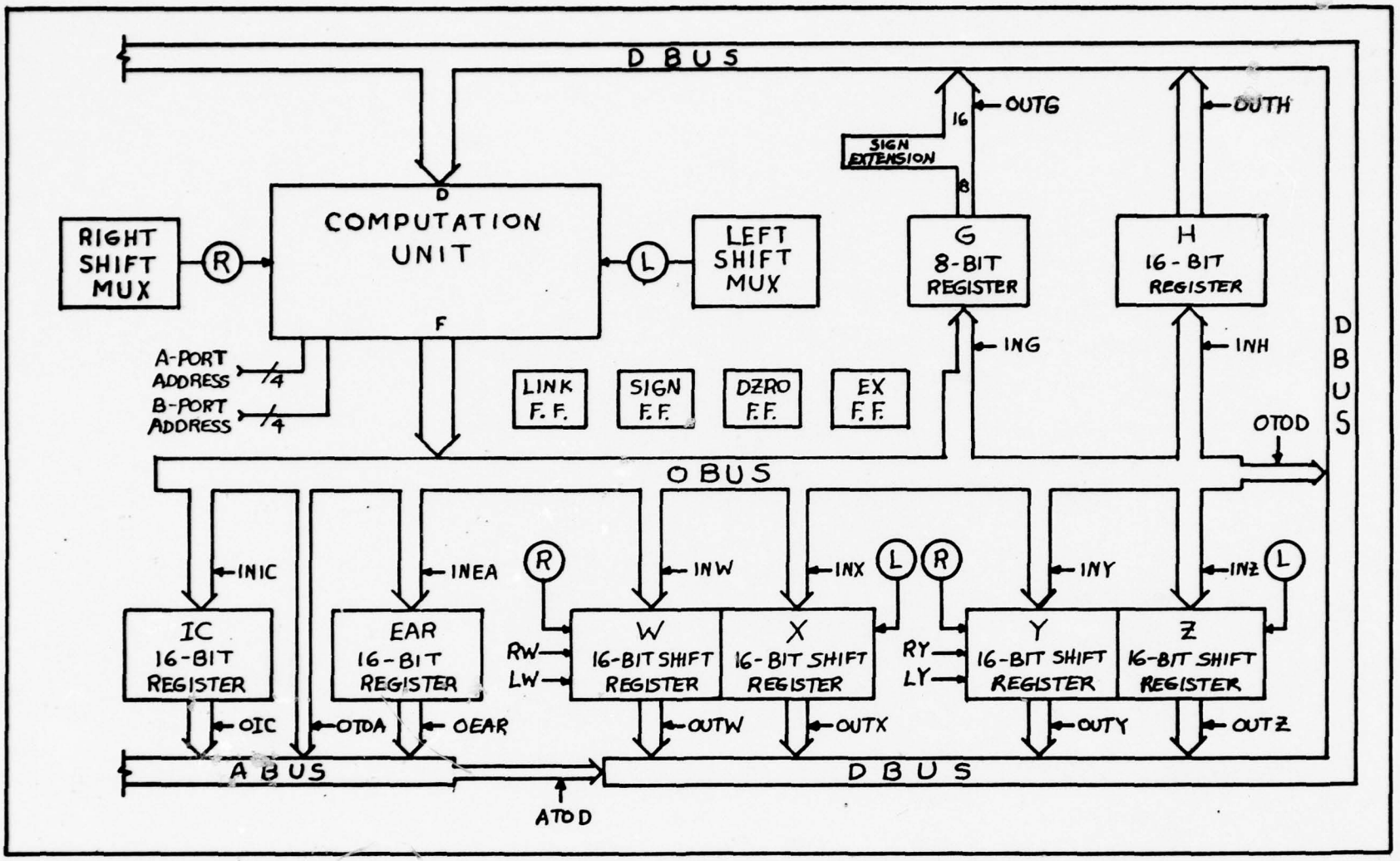


Fig. 8. Arithmetic and Logic Unit Block Diagram

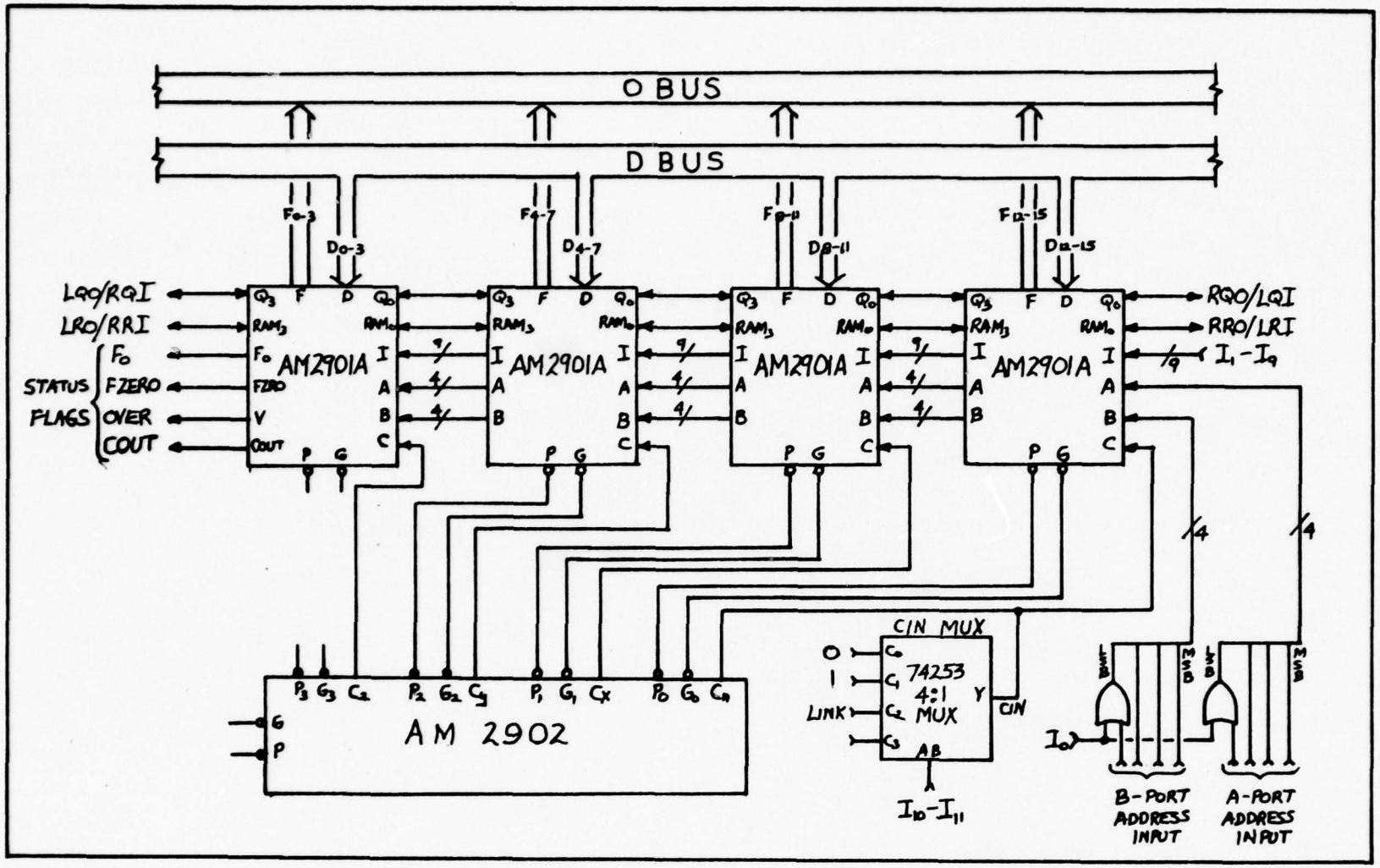


Fig. 9. ALU Computation Unit
Interconnection Diagram

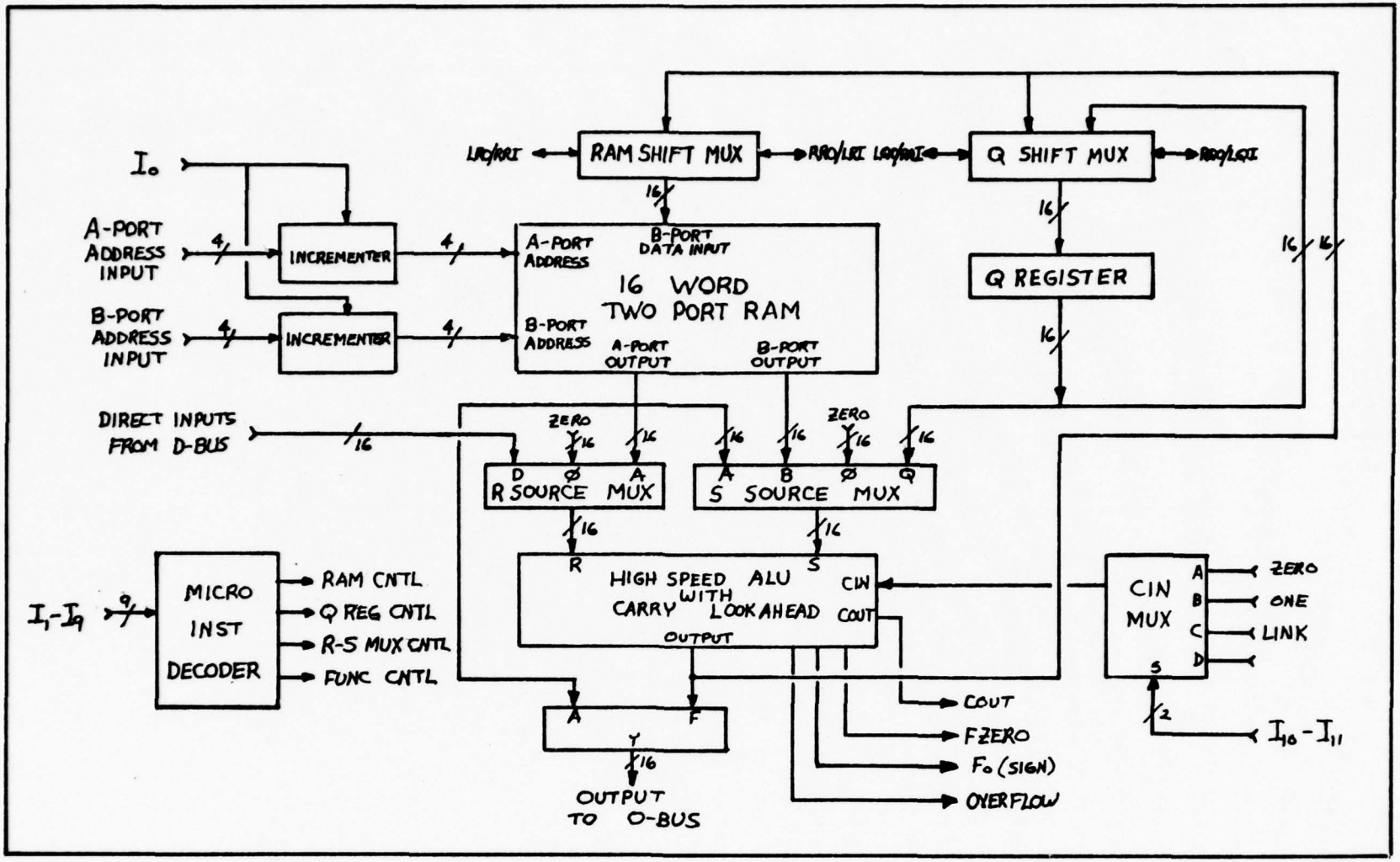


Fig. 10. ALU Computation Unit
Functional Representation

applying a 4-bit address to the B-address input. The same address can be applied to both the A-address input and the B-address input, in which case the same word will appear at the A-port and B-port outputs simultaneously.

The A-address and B-address inputs are driven by the A Latch and B Latch of the Computer Control Unit (described in Chapter V). The least significant bit of both the A-address and the B-address inputs are logically "OR'ed" with microcontrol signal I_0 (Figure 9). This "OR-ing" function is used to increment the RAM address inputs during double precision and floating point operations to address general purpose registers "RA+1" and "RB+1". The address fields can be incremented in this relatively simple manner because of the programming restriction that requires registers RA and RB to be even numbered registers when associated with two word operands.

Both output ports of the RAM are latched. When the clock input is "high", the A-port and B-port latches are open and will pass whatever data is present at the RAM outputs. When the clock input is "low", the latches are closed and will retain the last data entered.

The A-port of the RAM does not have a data input. Therefore, new data can only be written into the RAM through the B-port. This is accomplished by providing the new data to the B-port data input, providing a 4-bit address to the B-address input, and issuing the appropriate microcontrol signals. When the clock input is low, new data will be written into the RAM word (general purpose register) identified by the B-address input.

The B-port data input is driven by the Arithmetic Element output (F) through a shifting multiplexer. This shifter allows the AE output (F) to be shifted one bit to the right or left before it is written into the RAM word (general purpose register). To provide for a variety of shifting modes, the RAM shift multiplexer has two three-state linkage signals:

RIGHT RAM OUT/LEFT RAM IN (RRO/LRI) and LEFT RAM OUT/RIGHT RAM IN (LRO/RRI). The RRO/LRI signal line is used to output the "carry-out" bit of the RAM shifter during right shift operations and to input the "carry-in" bit during left shift operations. Likewise, the LRO/RRI signal line outputs the "carry-out" bit during left shift operations and accepts the "carry-in" bit during right shift operations.

Arithmetic Element. The Arithmetic Element (AE) can perform three binary arithmetic and five logical operations on two input words "R" and "S". These eight functions are controlled by the 3-bit microcontrol field (I_4 - I_6) defined on Table B2.

The "R" and "S" inputs of the AE are driven by multiplexers. The "R" input multiplexer has the RAM A-port (RA), the direct data input (D), and "Zero" (0) connected as inputs. Likewise the "S" input multiplexer has the RAM A-port (RA), the RAM B-port (RB), the Q register (A), and "Zero" (0) connected as inputs. This multiplexer scheme gives the capability of selecting various pairs of RA, RB, D, Q, and 0 inputs as source operands for the Arithmetic Element (AE). These five inputs, when taken two at a time, result in ten possible source operand pairs. Of the ten possible pairs of inputs, only eight are actually used. These eight operand pairs are selected by the 3-bit microcontrol field (I_1 - I_3). The RAM address incrementing function associated with microcontrol bit I_0 (previously described) augments the ALU source field (I_1 - I_3). Table B1 shows the ALU source operand pairs selected by the composite 4-bit microcontrol field (I_0 - I_3).

The two source operands not fully described as yet are the D input and the Q input. The D input is a 16-bit wide direct data field which is driven from the processor's D-BUS and is used to input all data into the

Computation Unit. The Q-register functions as a 16-bit multipurpose register which can be used for both scratch pad and shifting operations. The input of the Q register is driven by the Arithmetic Element output (F). The Q register performs three basic functions: it can load new data from the AE output (F), it can shift its contents one bit right, or it can shift its contents one bit left. These three functions are controlled by the 3-bit ALU destination microcontrol field (described below).

To provide for a variety of shift modes, the Q register has two three-state shift linkage lines: RIGHT Q OUT/LEFT Q IN (RQO/LQI) and LEFT Q OUT/RIGHT Q IN (LQO/RQI). When a right shift is performed, the RQO/LQI line outputs the carry-out bit from the LSB of the Q register and the LQO/RQI line accepts the bit to be shifted into the MSB of the Q register. Likewise, when a left shift is performed, the LQO/RQI line outputs the carry-out bit from the MSB of the Q register and the RQO/LQI line accepts the carry-in bit for the LSB of the Q register.

The Arithmetic Element output (F) can be routed to a number of destinations as indicated by Table B3. The 3-bit microcontrol field (I_7 - I_9) which controls the destination also controls the shift operations within the Computation Unit (CU). With exception of the destination function "BOA", the operations described in Table B3 are relatively straight forward and require little explanation. The AE output (F) is enabled onto the O-BUS and may also be routed to the data input of the RAM shifter and/or the input of the Q register shifter. The "BOA" destination function writes the AE output (F) into the RAM word defined by the B-address input in the normal manner; but, instead of enabling F onto the O-BUS, the output of the RAM A-port is routed directly onto the O-BUS.

The status of the Arithmetic Element (AE) is monitored through three status flags: "FO", "FZERO", and "OVR". The "FO" flag is derived from the most significant bit of the AE output (F) and corresponds to the sign bit of the output (F). A zero detector on the AE's output (F) transmits a "high" over the "FZERO" status line when F is equal to zero and a "low" when F is not equal to zero. The overflow flag (OVR) is "high", indicating an arithmetic overflow, whenever the carry in and carry out of the most significant bit of the AE are not equal.

Support Hardware

The shift multiplexers, carry-in multiplexer, and ALU flip flops, shown on Figure 8, directly support the operation of the Computation Unit (CU). These support devices are described in the following paragraphs.

Shift Multiplexers. The ALU contains two shift linkage multiplexers which generate a variety of linkages for different shift operations. The Right Shift Multiplexer (R-MUX) provides the carry-in signal for all right shift operations and the Left Shift Multiplexer (L-MUX) generates the carry-in signals for left shift operations. A block diagram of these multiplexers is shown on Figure 11.

The shift linkage multiplexers drive the three-state LRO/RRI and RQO/LQI linkage lines of the Computation Unit (CU) through three-state buffers which are controlled by microcontrol bit I_8 of the ALU destination field. Microcontrol bit I_8 is always "false" during right shift operations of the CU and is always "true" during CU left shift operations. When I_8 is "false", the output of the R-MUX drives the LRO/RRI line and the signal from the L-MUX to the RQO/LQI line is disabled. Similarly, when I_8 is "true", the output of the L-MUX is passed to the RQO/LQI line and the R-MUX output

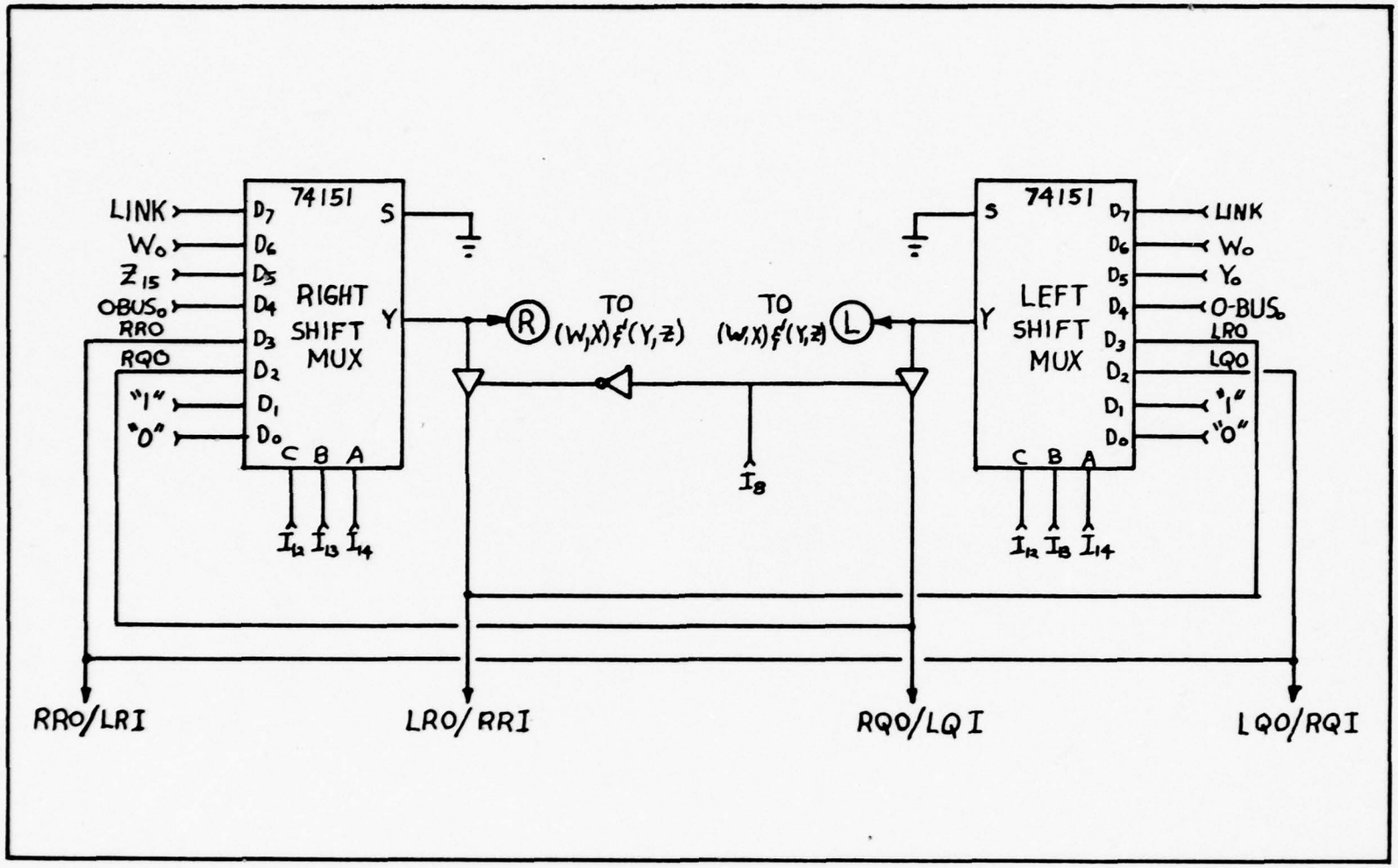


Fig. 11. ALU Shift Linkage Multiplexers

is blocked from the LRO/RRI line.

The LRO/RRI and RQO/LQI linkage lines of the Computation Unit (CU) are tied together as shown on Figure 11. This linkage arrangement cascades the shift operation of the RAM shifter and the Q register shifter to form a 32-bit shift unit for two word (long) shift operations. Although this arrangement facilitates two word (long) shift operations and reduces shift linkage complexity, it imposes a microprogramming restriction. Single word (short) right shift operations of the CU must be performed in RB (RAM shifter) and single word (short) left shift operations must be performed in the Q register.

In addition to the supporting shift operations of the Computation Unit, these shift multiplexers also provide the shift linkage signals for the ALU's shift register pairs (WX and YZ) which will be described later in this chapter. This common use of the shift linkage multiplexers by three different shift units (CU, register pair WX, and register pair YZ) simplifies processor hardware; but, it imposes another microprogramming restriction. Parallel operation of these three shift units is possible only if the shift operations do not require conflicting shift linkages.

Figure 11 shows the eight shift linkage sources which are available to the multiplexer inputs. The 3-bit microcontrol field (I_{12} - I_{14}) which controls the multiplexers' operation is described on Table B5.

Carry-In Multiplexer. The arithmetic functions of the Computation Unit (ADD, S-R, and R-S) are affected by the carry-in (CIN) signal. The CIN signal facilitates double precision (two word) operations by accepting the linkage signal passed from the least significant word to the most significant word and it controls the arithmetic mode (1's complement or 2's complement). To illustrate how the carry-in value controls the arithmetic mode, consider

the subtract functions "S-R". With a "zero" carry-in value, the function "S-R" executes a 1's complement subtraction. However, the same function "S-R" performs a 2's complement subtraction when a "one" carry-in value is applied.

The Carry-in Multiplexer selects the carry-in signal from one of three potential sources (Zero, One, and Link). Table B4 defines the 2-bit microcontrol field (I_{10} - I_{11}) which controls the operation of the Carry-In Multiplexer.

ALU Flip Flops. Support hardware includes four flip flops (SIGN, LINK, DZRO and EX). Figure 12 shows how these flip flops are connected. The operation of these flip flops is described below.

SIGN Flip Flop. The SIGN flip flop is used to save the sign of the result for multiplication and division. This allows all operands to be forced positive, (to simplify calculations) and to then assign the correct sign to result, upon completion of the operation. The SIGN flip flop operates under control of the microcommand field (I_{26} - I_{30}) which is defined on Table B9. The microcommand "PCLR" causes the SIGN flip flop to set its output to zero and the microcommand "ISGN" causes the flip flop to invert its output. The output of the SIGN flip flop is routed to the condition test multiplexer in the Computer Control Unit (Chapter V).

LINK Flip Flop. The primary purpose of the LINK flip flop is to provide linkage between the first and second word to two word (double precision/floating point) arithmetic operations. To accomplish that purpose, the output of the LINK flip flop is routed to an input of the Carry-In Multiplexer (previously described). The flip flop's output is also routed to the condition test multiplexer in the Computer Control Unit where it is used to detect arithmetic overflow during multiplication and division routines.

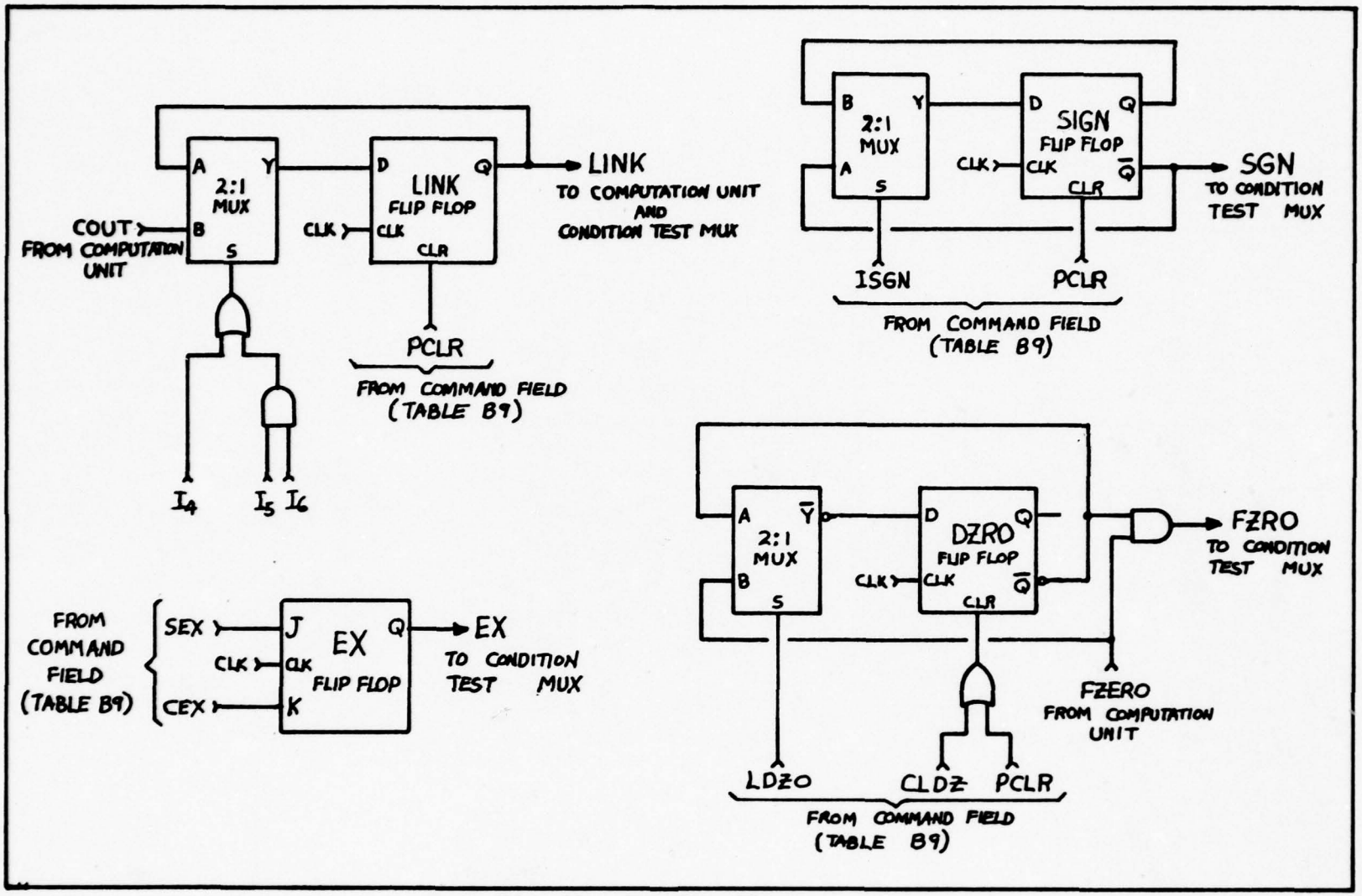


Fig. 12. ALU Flip Flops

The LINK flip flop automatically saves the bit carried out (COUT) of the most significant bit of the Arithmetic Element (AE) during arithmetic operations (ADD, S-R, and R-S). When the AE is not performing an arithmetic operation, the previous link value is saved. The LINK flip flop is cleared by the microcommand "PCLR" defined on Table B9.

DZRO Flip Flop. The zero detector on the output of the Arithmetic Element (AE) which generates the DZERO flag can only inspect 16 bits at a time and cannot determine if a two word operand is equal to zero. The DZRO flip flop is used to circumvent this problem. The output of the DZRO flip flop controls the gate which transmits the FZERO flag from the AE to the condition test multiplexer in the Computer Control Unit. When the DZRO output is "high" the FZERO flag is passed directly to the FZRO input of the condition test multiplexer. When the DZRO output is "low" the FZERO flag is blocked and a "false" is sent to the condition test multiplexer which indicates that the result is not equal to zero.

The DZRO flip flop operates under direct microprogram control The "PCLR" and "CLDZ" microcommands issued by the command field (I_{26} - I_{30}), shown on Table B9, set the DZRO output "high". The DZRO flip flop is loaded with the current value of the FZERO flag when the "LDZ" microcommand is generated by the command field (I_{26} - I_{30}). The "LDZ" microcommand is normally issued when the first word of a two word operand is being processed by the Arithmetic Element. It causes the flip to be set if the word was equal to zero and cleared otherwise. When the second word of the two word operand is processed, the FZRO condition test will be "true" only if both halves of the two word operand were equal to zero.

EX Flip Flop. The Execute (EX) macroinstruction presents a unique program control problem. When the EX instruction is executed, it causes the macroinstruction pointed to by its derived address to be executed. If that new instruction causes a jump, then execution will continue from the jump location. If the new instruction does not cause a jump, then execution will continue from the next sequential instruction following the EX instruction.

The EX flip flop is used by microprogram software to implement the Execution (EX) instruction. Each time the EX instruction is executed, the microprogram will set this flip flop using the "SEX" microinstruction (Table B9). Whenever a jump is performed the microprogram will clear this flip flop by issuing the "CEX" microinstruction (Table B9). The state of this flip flow is tested at the end of each macroinstruction cycle. If the flip flop is set, the microprogram will assume that the previous instruction was executed from the derived address of an EX instruction; and the microprogram will return program control to the next sequential instruction following the EX instruction.

The output of this flip flop is routed to the Condition Test Multiplexer in the Computer Control Unit and is tested by the "EX" condition test (Table B14). The output of the EX flip flop is also OR'ed with the Interrupt Flag from the Interrupt Control Unit. This reduces the overhead required to test the state of the flip flop by allowing the microprogram to test the EX flip flop at the same time it tests for interrupts.

External Registers

In addition to the Computation Unit and its supporting hardware, the ALU includes a variety of external registers. These registers (W, X, Y, Z, G, H, IC, and EAR), shown on Figure 8, can be classified into three categories according to their function.

Shift Register Pairs. Registers "W", "X", "Y", and "Z" are each 16-bit general purpose shift registers which receive their parallel inputs from the O-BUS and enable their three-state parallel outputs onto the D-BUS. Although each of these registers performs its input/output function independently, the shifting function of register pair "W" and "X" and register pair "Y" and "Z" are cascaded to form two 32-bit shift registers.

The main function of the shift register pairs is to increase throughput by facilitating the shifting operations associated with multiplication and division. The external shift control field (I_{15} - I_{17}), described in Table B6, controls the shifting functions of these shift register pairs. Shift linkages for these registers are provided by the same shift linkage multiplexers (R-MUX and L-MUX) which support the shift operations of the Computation Unit.

Exponent Storage. Register "G" is used to store the 8-bit exponents of floating point operands. When this 8-bit register is loaded from the O-BUS, the least significant 8-bits of the O-BUS are latched and the most significant 8-bits are lost. The output of register "G" is connected to the D-BUS through three-state buffers and drives the least significant 8-bits of the bus. Sign extension hardware extends the most significant bit of the output to the most significant 8-bits of the D-BUS.

Program Control. The Instruction Counter (IC) and the Effective Address Register (EAR) are concerned with controlling the sequence in which instructions and operands are fetched from memory. These 16-bit registers receive their input from the O-BUS and enable their three-state outputs onto the A-BUS.

Register H is used to save the original contents of the Instruction Counter when the Execute (EX) macroinstruction is executed. This 16-bit register receives its input from the O-BUS and enables its three-state outputs onto the D-BUS.

All working registers described above are loaded under control of the Register in field (I_{22} - I_{25}) of the microinstruction word as shown in Table B8. The three-state output enables signals of the shift register pairs, the G register, and the H register are controlled by the Register Out Field (I_{18} - I_{21}) of the microinstruction word shown on Table B7. The output enable signals of the IC and EAR are controlled by the "OEAR" and "OIC" commands generated by the Command field (I_{26} - I_{30}) which is shown on Table B9.

Summary

This chapter presented the functional design of the processor's Arithmetic and Logic Unit (ALU). It described the ALU's basic architecture, its functional components, and its response to specific microinstruction signals.

In the next three chapters the remaining components of the processor will be described; Chapter V describes the Computer Control Unit; Chapter VI describes the Interrupt Control Unit; and Chapter VII describes the Input/Output Interface Unit.

V. Computer Control Unit Design

The Computer Control Unit (CCU) has three basic responsibilities. First, the CCU is responsible for controlling the sequence in which microinstructions are fetched from its control memory. Second, the CCU must decode the operand and data fields of the machine instructions according to the instruction's format and route the information to the proper destination. Third, the Computer Control Unit is responsible for monitoring the processors's status flags and maintaining the processor's status register. This chapter will describe the design of the CCU in three sections. Each of these sections will describe one aspect of the CCU's responsibilities.

Microprogram Sequencing

The basic architecture of the Computer Control Unit (CCU), shown on Figure 13, is typical of most microprogrammed control units. The control signals which govern the processor's operation are encoded into microinstruction words which are stored in the CCU's control memory. When a new machine instruction is selected for execution, the CCU interprets the instruction's operation code and then outputs an appropriate sequence of microinstructions to execute the intended operation.

Control Memory. The CCU's control memory is 60 bits wide and stores up to 4096 microinstruction words. The address inputs of the control memory are driven by the output of the AM 2910 microsequencer. The output of the control memory is latched into the pipeline register on the rising edge of each clock pulse.

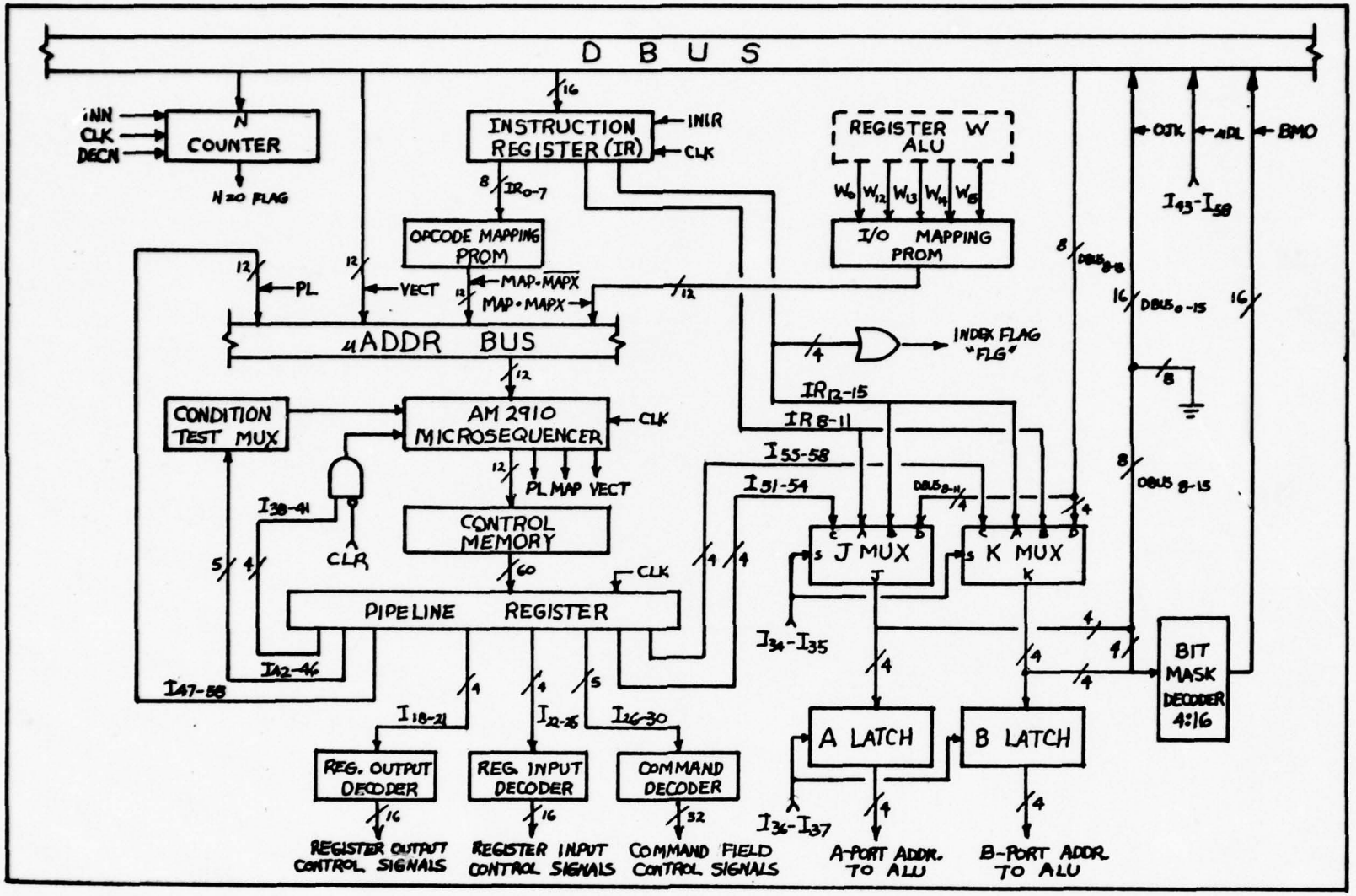


Fig. 13. Computer Control Unit Block Diagram

Pipeline Register. The 60 bit pipeline register contains the microinstruction currently being executed. Latching microinstructions in this register allows the fetching cycle for the next microinstruction to be overlapped with the execution of the current microinstruction to increase throughput. The 60 bit microinstruction is divided into fields as shown in Figure 14. These fields are summarized below.

Pipeline Bits	Description	Reference
I0 - I3	ALU Source Control	Table B1
I4 - I6	ALU Function Control	Table B2
I7 - I9	ALU Destination Control	Table B3
I10 - I11	ALU Carry-In Control	Table B4
I12 - I14	ALU Shift MUX Control	Table B5
I15 - I17	External Shift Control	Table B6
I18 - I21	Register Output Control	Table B7
I22 - I25	Register Input Control	Table B8
I26 - I30	Command Field (Miscellaneous)	Table B9
I31 - I33	Input/Output Interface Control	Table B10
I34 - I35	JK MUX Control	Table B11
I36 - I37	AB Latch Control	Table B12
I38 - I41	Next Address Control	Table B13
I42 - I46	Condition Test MUX Control	Table B14
I47 - I50	Interrupt Control	Table B15
I47 - I58	Direct Address	- -
I51 - I58	Direct Data Short	- -
I43 - I58	Direct Data Long	- -
159	SPARE	- -

The Interrupt Control, Direct Address, Direct Data Short, and Direct Data Long fields are overlapped. Steering of these fields is accomplished indirectly. For example: microinstruction bits (I47 - I50) are interpreted as an interrupt command only if the Interrupt Instruction Enable "INTE" command is issued by the command field; interpretation of bits (I47 - I58) as Direct Data Short field is dependent upon the command given to the JK multiplexer; and interpretation of bits (I43 - I58) as a Direct Data Long field is controlled by the "uDL" command from the Register Output field (I18 - I21).

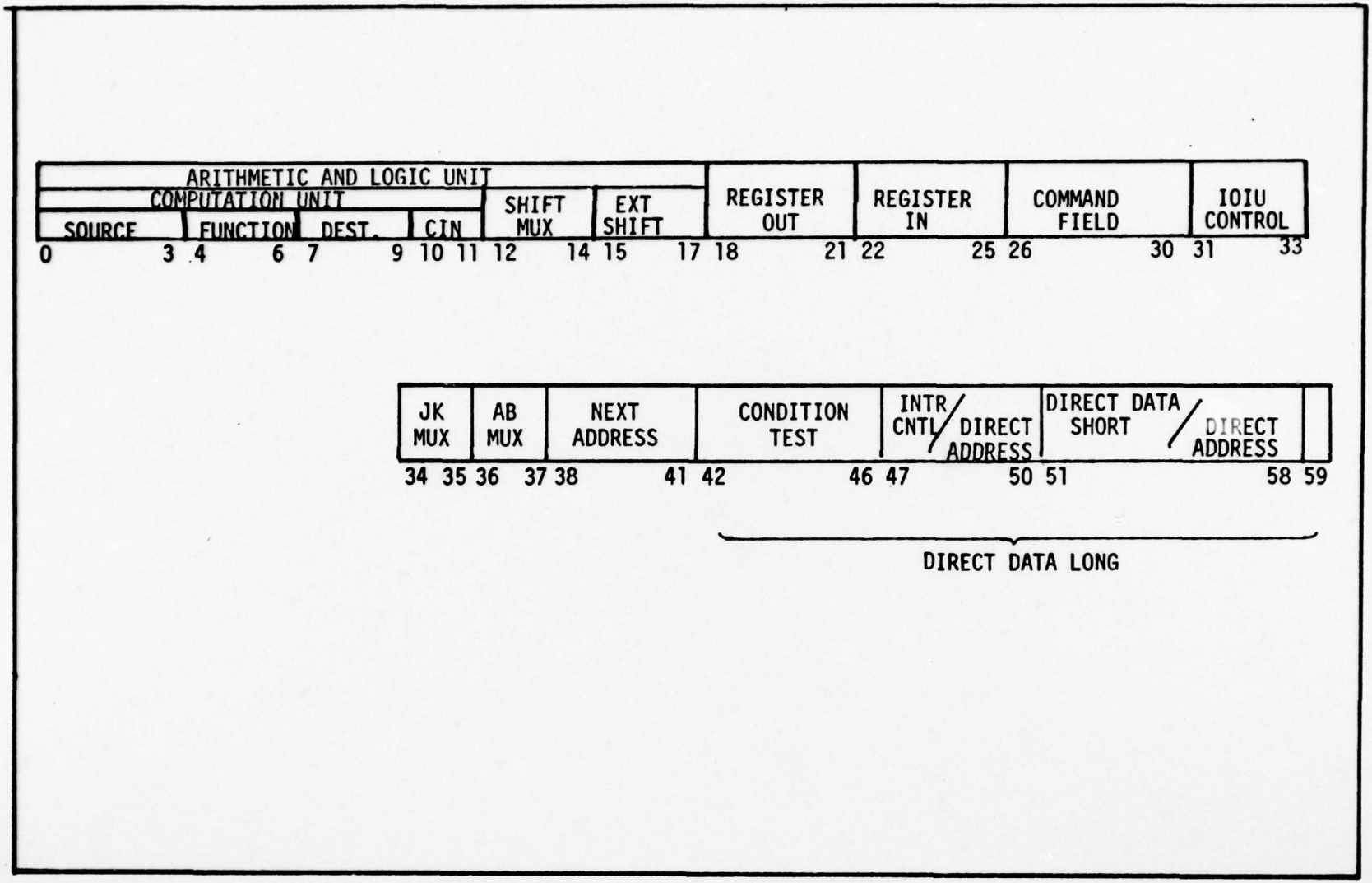


Fig. 14. Microinstruction Word Format

This indirect control of format imposes a number of microprogramming restrictions. For example, an interrupt command and a conditional branch to a direct address cannot be executed during the same microcycle. These restrictions will become more apparent when the processor's microroutines are presented in Chapter VIII.

Microsequencer. The key element of the CCU is an AM 2910 Microprogram Controller (microsequencer). In simple terms, the function of this microsequencer is to select the control memory address of the next microinstruction from a variety of sources. The AM 2910 block diagram presented on Figure 15 shows three sources for this next address, which are internal to the AM 2910, and one input port for external sources.

Internal Microaddress Sources. The three internal next address sources in the AM 2910 microsequencer are: the register/counter, the internal stack, and the microprogram counter. These three internal sources are described in the following paragraphs.

The register/counter is a 12-bit wide counter which is loaded and/or decremented under control of the 4-bit Next Address Control field (Table B13) which will be described later in this chapter. The output of this counter is routed to both the "R" input of the AM 2910's internal multiplexer and a zero detector. The use of this register/counter in counting microprogram loops and performing three-way-branches will be described when the sixteen operations of the microsequencer are discussed.

The five deep internal push down stack of the AM 2910 also operates under the control of the Next Address Control field (I38 - I40). This stack allows the microsequencer to link up to five levels of nested microsubroutines.

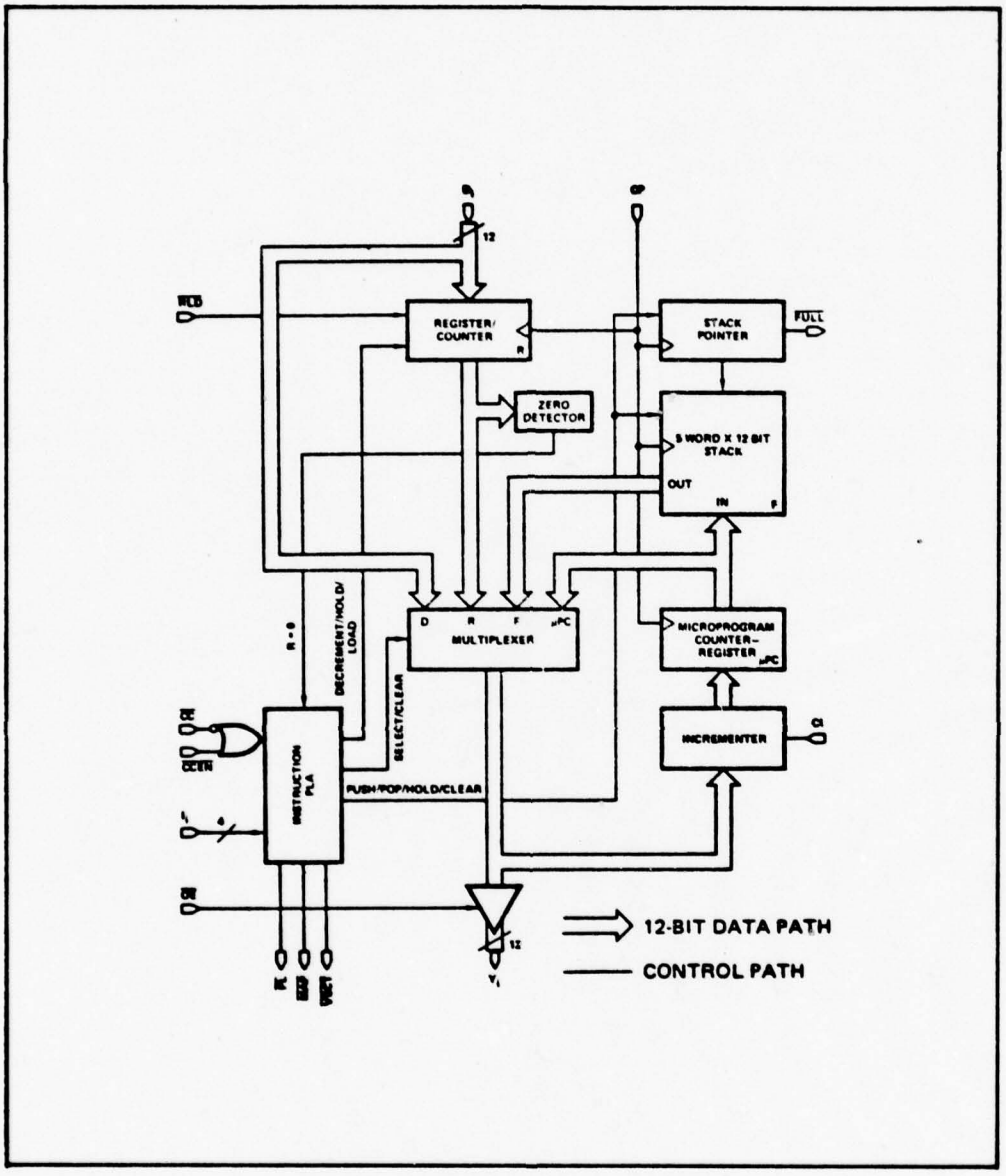


Fig. 15. AM 2910 Microsequencer
Block Diagram
(From Ref 4:1)

During each microcycle, the microprogram counter is automatically loaded with the current microaddress incremented by one. This register provides a means of sequentially stepping through a contigious sequence of microinstructions.

External Microaddress Sources. The direct input port (D) of the AM 2910 microsequencer receives its input from the 12-bit wide, three-state uADDR BUS shown on Figure 13. The uADDR BUS may be driven by either the OP Code Mapping PROM, I/O Mapping PROM, D-BUS, or Pipeline Register. The three-state outputs of these four external sources are controlled by the "PL", "VECT", "MAP" and "MAPX" control signals. The "PL", "VECT", and "MAP" control signals are generated automatically by the AM 2910 in response to the Next Address control field. The "MAPX" control signal is issued manually by the Command field (I26 - I30) which is shown on Table B9. These four external sources are described in the following paragraphs.

The OP Code Mapping PROM (MAP) accepts the 8-bit OP Code from the most significant half of the instruction register (IR) and maps it into a 12-bit microaddress. This microaddress points to the starting address of the microroutine which implements the operation specified by the OP Code. Its three-state output is enabled on to the uADDR BUS when the AM 2910 activates the MAP control line and the "MAPX" control line is "false".

The I/O Mapping PROM (I/O MAP) performs a function similar to the OP Code Mapping PROM. The second word of two word Input/Output instructions identifies the specific operation to be performed and can be viewed as a 16-bit OP Code extension "OXC". When a Input/Output instruction is executed, the OXC is loaded into register W of the ALU. The I/O Map inspects 5 bits (W0, W12, W13, W14, and W15) of register W and maps these bits into a 12-bit microaddress for specific I/O operations as shown on Table VI. The

TABLE VI

I/O Mapping PROM

OXC	PROM Inputs from Register W					I/O OPERATION
	W0	W12	W13	W14	W15	
4000	0	0	0	0	0	Trigger GO/NO GO Indicator
4001	0	0	0	0	1	Output to TIMER A
4002	0	0	0	1	0	Output to TIMER B
4003	0	0	0	1	1	SPARE
4004	0	0	1	0	0	SPARE
4005	0	0	1	0	1	DMA Enable
4006	0	0	1	1	0	DMA Disable
4007	0	0	1	1	1	SPARE
4008	0	1	0	0	0	Interrupt Disable
4009	0	1	0	0	1	Output to Discrete I/O Channel
400A	0	1	0	1	0	Clear Interrupts
400B	0	1	0	1	1	Set Interrupts
400C	0	1	1	0	0	Interrupt Enable
400D	0	1	1	0	1	SPARE
400E	0	1	1	1	0	SPARE
400F	0	1	1	1	1	SPARE
C000	1	0	0	0	0	SPARE
C001	1	0	0	0	1	Input from TIMER A
C002	1	0	0	1	0	Input from TIMER B
C003	1	0	0	1	1	Input from STATUS WORD
C004	1	0	1	0	0	SPARE
C005	1	0	1	0	1	SPARE
C006	1	0	1	1	0	SPARE
C007	1	0	1	1	1	SPARE
C008	1	1	0	0	0	SPARE
C009	1	1	0	0	1	Input from Discrete I/O Channel
C00A	1	1	0	1	0	SPARE
C00B	1	1	0	1	1	SPARE
C00C	1	1	1	0	0	SPARE
C00D	1	1	1	0	1	SPARE
C00E	1	1	1	1	0	SPARE
C00F	1	1	1	1	1	SPARE

three-state outputs of the I/O Map are enabled on to the uADDR BUS when the "MAP" control line and the "MAPX" control line are both "true".

The 12-bit Direct Address field (I47 - I58) of the microinstruction pipeline register can be enabled on to the uADDR BUS to provide the address for microinstruction branches. The "PL" signal from the AM 2910 enables this three-state direct address on to the uADDR BUS.

The least significant 12 bits of the D-BUS (D4 - D15) can also drive the uADDR BUS. The input path is typically used to load data into the AM 2910's internal register/counter. The D-BUS (D4 - D15) is enabled on to the uADDR BUS when the AM 2910 activates the "VECT" control line under control of the Next Address field (I38 - I41).

Condition Test Multiplexer. The Condition Test Multiplexer selects one of 16 signals as the test input for conditional branching instruction. The block diagram of the Condition Test Multiplexer is shown on Figure 16. Table B14 shows the 5-bit Microcontrol field (I42 - I46) which controls the operation of this multiplexer. Bits (I43 - I46) actually select one of the 16 possible tests. Microcontrol bit I43 is a polarity control which has the capability to invert the multiplexer's output.

Microsequencer Operations. The operation of the Microsequencer is controlled by the 4-bit Next Address field (I38 - I44) of the microinstructions and the condition test signal from the condition test multiplexer. The Next Address field (Table B13) directs the microsequencer to execute one of sixteen different operations. A description of these operations is presented in the following paragraphs.

(1) Jump Zero. The Jump Zero (JZ) instruction forces the microprogram location counter to address Zero. The microcode for this instruction is automatically jammed onto the 4-bit Next Address Control

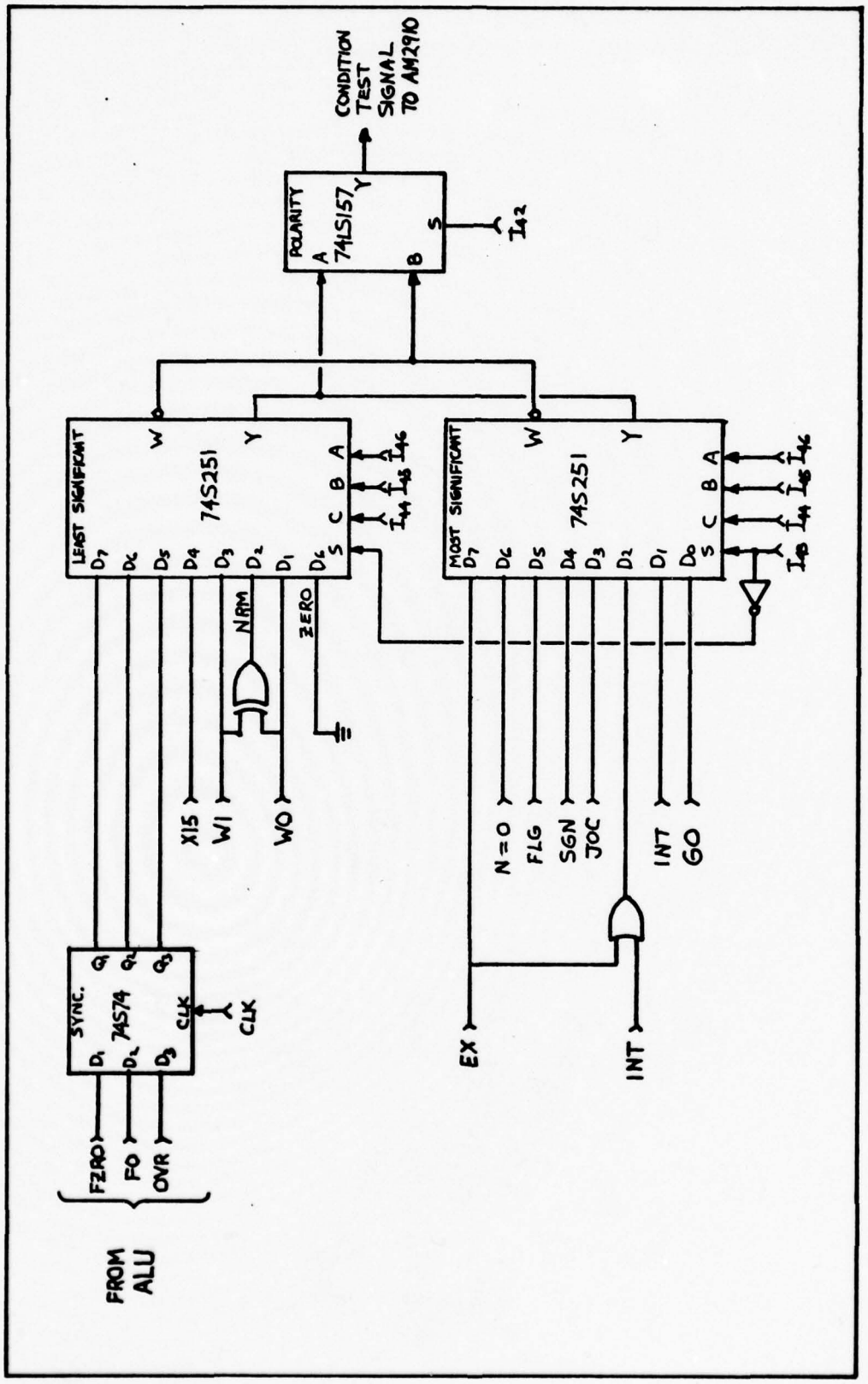


Fig. 16. Condition Test Multiplexer Block Diagram

Lines of the AM 2910 by hardware (AND gates) whenever the processor's master clear line (CLR) is "true". This instruction is used to reset the microsequencer to location Zero which is reserved for the beginning of the Power-Up Microroutine.

(2) Conditional Jump-to-Subroutine. The Conditional Jump-to-Subroutine (CJS) instruction is used to execute conditional jumps to micro-subroutines using the address provided by the Direct Address field of the pipeline register (I47 - I58). If the condition test is passed (TEST input "high"), the current microprogram address is incremented and pushed on to the internal stack (to provide return linkage) and the next microinstruction executed will be fetched from the address proved by the Direct Address field (I47 - I58). If the condition test fails (TEST input "low"), jump to subroutines will not be performed, and the next sequential address will be provided by the microprogram counter.

(3) Jump Map. The Jump Map (JMAP) instruction causes the $\overline{\text{MAP}}$ control line to be enabled and selects one of the Mapping PROMS as the source of the next microinstruction address. If the "MAPX" command is issued by the Command field concurrently with this instruction, the I/O Mapping PROM will be selected; otherwise, the OP Code Mapping PROM will be selected.

(4) Conditional Jump Pipeline. The Conditional Jump Pipeline (CJP) derives its branch address from the Direct Address field (I47 - I58) of the pipeline register. If the conditional test passes, microcontrol passes to the address specified in the Direct Address field; otherwise the next sequential address is provided by the microprogram counter.

(5) PUSH/Conditional Load Counter. The PUSH/Conditional Load Counter (PUSH) instruction is intended primarily for setting up loops in

microprograms. This instruction pushes the next sequential instruction address on to the stack and inspects the condition test input. If the test is passed, the internal counter/register is loaded with information provided by the Direct Address field. Otherwise, the counter/register is not loaded.

(6) Conditional Jump-to-Subroutine. The Conditional Jump-to-Subroutine (JSRP) selects one of two subroutines for execution based on a test condition. The instruction always increments the current instruction address and pushes it on to the stack to provide return linkages. Then, if the test is passed, the microsequencer branches to the subroutine address furnished by the contents of the internal register/counter. Otherwise, the microsequencer branches to the subroutine address provided by the Direct Address field (I47 - I58).

(7) Conditional Jump Vector. The Conditional Jump Vector (CJV) provides the capability to select the branch address from the D-BUS (D4 - D15). If the condition test is passed, the next instruction address is obtained from the D-BUS; otherwise, the next sequential instruction is provided by the microprogram counter.

(8) Conditional Jump R/PL. The Conditional Jump via R/PL instruction (JRP) selects the next instruction address from one of two sources. If the condition test passes, the contents of the internal register/counter is selected as the source of the next address; otherwise, the Direct Address field is selected as the next address source.

(9) Repeat Loop. The Repeat Loop (RFCT) instruction makes use of the decrementing capabilities of the register/counter. To be useful, some previous instruction (such as PUSH) must have loaded the counter/register. This instruction checks to see if the register/counter contains

a non-zero value. If so, the register/counter is decremented, and the address of the next microinstruction is taken from the top of the internal stack. If the register/counter contains zero, the loop is exited and control falls through to the next sequential microinstruction. The stack is then POP'ed by decrementing the stack pointer, but contents at the top of the stack are thrown away.

(10) Repeat Pipeline Register, Counter ≠ Zero. The Repeat Pipeline Register instruction (RPCT) is similar to the RFCT instruction (described above) except that the branch address now comes from the pipeline register Direct Address field rather than the stack.

(11) Conditional Return-from-Subroutine. As the name implies, the Conditional Return-from-Subroutine (CRTN) is used to branch from a subroutine back the next microinstruction address following the subroutine call. If the condition test is passed, the return address is POP'ed from the top of the internal stack and control transfers to the return address. If the test fails, the return is ignored and control falls to the next sequential microinstruction.

(12) Conditional Jump Pipeline and POP. The Conditional Jump Pipeline register and POP stack instruction (CJPP) provides a technique for loop termination and stack maintenance. If the test is passed, control will be passed to the address provided by the Direct Address field (I47 - I58) and the stack will be POP'ed (the value POP'ed off the stack is lost). If the test fails, the instruction is ignored and control falls to the next sequential microinstruction.

(13) Load Counter and Continue. The Load Counter and Continue (LDCT) instruction allows the counter to be loaded with the contents of the Direct Address field. This instruction always selects the next

sequential address as the next microinstruction address.

(14) Test End-of-Loop. The capability for conditionally exiting at the bottom of the loop is provided by the Test End-of-Loop (LOOP) instruction. If the test condition is "false", the instruction will cause microprogram control to branch to the address at the top of the internal stack. If the test is "true", program control falls through to the next sequential address.

(15) Continue. The simplest next address instruction is Continue (CONT). This instruction causes the program counter to increment so that the next sequential address is selected.

(16) Three-Way Branch. The most complex next address instruction is the Three-Way Branch (TWB). This instruction uses both the condition test input and the counter/register zero dectector to select one of three microinstruction addresses as the next microinstruction to be performed. For proper use of this instruction, the counter/register should have been loaded previously. As long as the condition test input is "false", the "TWB" branch instruction performs a decrement and branch until zero function, similar to the "RFCT" instruction. The next address is taken from the top of the stack until the count reaches zero; then the next address comes from the Direct Address field (I47 - I58). If the condition test is passed, control falls through to the next sequential microinstruction address. When the loop is ended, by either a zero count or by passing the conditional test, the value at the top of the stack is POP'ed and thrown away.

N-Counter. The AM 2910's internal counter/register can be used as a loop counter to implement loops in microinstruction routines. However, to increase the processor's throughput, a separate N-Counter will be used

as the loop counter in this design. Using the AM 2910's register/counter as a loop counter has several drawbacks. First, the uses of the register/counter as either a microinstruction register or loop counter are mutually exclusive. This limits flexibility. Second, switching the mode of the register/counter operation from register to loop counter requires a certain amount of lead time and overhead to load the register/counter with the proper information. Finally, the register/counter is controlled by the Next Address field which is also responsible for selecting the next microinstruction address. This dual use of the Next Address field causes occasional conflicts which waste microcycles. These wasted microcycles are a serious problem in the execution of multiplication and division routines. Providing a separate N-Counter to control microroutine loops eliminates these problems and allows the register/counter to be reserved for use as a microprogram register.

The N-Counter is a 16-bit down counter. It is loaded from the D-BUS when the "INN" microcommand is issued by the Register In field (Table B8). The contents of the N-Counter are decremented whenever the "DECN" microcommand is issued by the Command field (Table B9).

The output of the N-Counter is routed to a zero detector which drives the "N = 0" input of the condition test multiplexer. If the contents of the N-Counter are equal to zero, the "N = 0" test will be "true", otherwise, the "N = 0" test will be "false".

Instruction Decoding

The second function of the Computer Control Unit is to decode the data and operand fields of machine instruction and route the information to the proper destination. Decoding the information fields of a variety

of machine instruction formats is accomplished through two multiplexers, the J Multiplexer (JMUX) and the K Multiplexer (KMUX).

J-K Multiplexers. These input multiplexers are each 4 bits wide. The input sources for the JMUX are the Pipeline Register (I51 - I54), Instruction Register bits (IR8 - IR11), Instruction Register bits (IR12 - IR15), and Data BUS (D8 - D11). The source inputs for the KMUX are the Pipeline Register (I55 - I58), Instruction Register bits (IR12 - IR15), Instruction Register bits (IR8 - IR11), and Data BUS (D12 - D15). Selection of the source is controlled by the JK Multiplexer field (I34 - I55) as shown on Table B11.

The output of the J and K Multiplexers are referred to as "J" and "K" respectively. The "J" output is routed to two sources: the A latch and the D-BUS. The "K" output is routed to three sources: The B latch, the Bit Mask Decoder, and the D-BUS.

AB Latches. The A and B latches are each 4-bit registers which latch the A Port address and the B Port address, and drive the A Port and B Port address inputs of the Computational Unit's two port RAM. These latches are controlled by the AB Latch Control field (I36 - I37) of the microinstruction (Table B12).

The A Port and B Port addresses are selected by the JK multiplexers at the start of a machine instruction and then latched in the A and B latches. This frees the JK Multiplexers for other uses during the instruction execution. These other uses will become more apparent when the microroutines are described in Chapter VIII.

Bit Mask Decoder. The Bit Mask Decoder is essentially a 1 of 16 decoder. It decodes the 4-bit K-field value N into a 16 bit word with bit N set to one and all other bits set to zero. The three-state output

of this decoder is enabled on to the D-BUS when the "BMO" microinstruction is issued (Table B7). This bit mask is used to execute machine instructions such as Set Bit (SB, SBR, SBI), Reset Bit (RB, RBR, RBI), and Test Bit TB, TBR, TBI).

Status Register

The third function of the Computer Control Unit is to monitor the processor's status and maintain the processor's status register. Figure 17 shows the hardware associated with this function.

The status register can be loaded directly from the D-BUS or it can be set by the status flags of the processor. Loading the register directly from the D-BUS is accomplished by issuing the "INS" command of the Register In field (Table B8). Setting the status register from the processor status flags can be accomplished by four different commands from the Command field (Table B9). The "STI" command loads the status register with the current state of the status flags. The "STU", STV", and "STC" commands are similar to the basic "STI" command except that they set one of the status flags before loading the register. The "STC" command loads status with the illegal OP Code flag set; "STU" loads status with the Underflow flag set; and "STV" loads status with the Overflow flag set.

The contents of the status register can be output onto the D-BUS by issuing the "OUTS" command of the Register Out field (Table B7). This feature is used to save the processor's status during interrupts.

The first four bits of the status register (Sign, Zero, OV, and UV) are routed through combination logic elements which generate the 4-bit Condition Status (CS) field defined below:

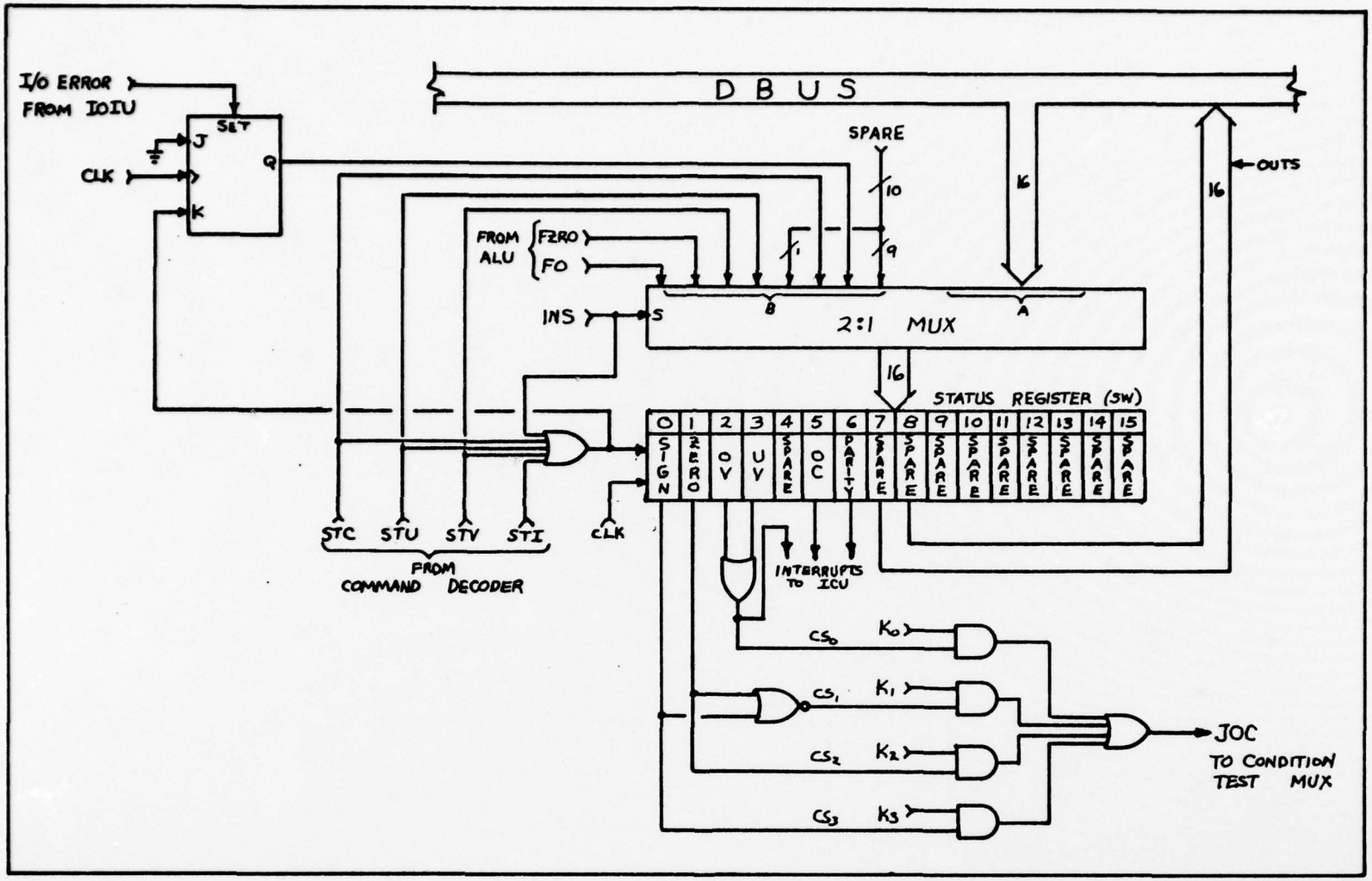


Fig. 17. Status Register and Condition Status Block Diagram.

CS_0 ← 1 if status overflow or underflow

CS_1 ← 1 if status greater than zero

CS_2 ← 1 if status equals zero

CS_3 ← 1 if status less than zero

The 4-bit Condition Status field is bit-by-bit AND'ed with the 4-bit Condition (C) field (IR8 - IR11) of the Jump on Condition (JC or JCI) machine instruction and the resulting 4 bits are OR'ed together. This logical operation allows the 4-bit Condition field to specify 16 possible Jump Conditions. The output of the final OR gate is labeled "JOC" and it is routed to one of the inputs of the condition test multiplexer. The JOC line is "true" if the contents of the status register matches the condition specified by the C field.

Summary

This chapter presented the functional design of the processor's Conputer Control Unit (CCU). It described the CCU's functions, its operation, and its components, as well as the format of the micro-instruction word.

This chapter did not include a description of the processor machine states or its microinstruction routines. Those topics will be discussed in Chapter VIII, after the Interrupt Control and Input/Output Interface Units have been discribed.

VI. Interrupt Control Unit Design

The interrupt system, described in Chapter II, is managed by the Interrupt Control Unit (ICU). The ICU accepts interrupt requests from sixteen sources and latches those requests in the Interrupt Pending Register. It masks the latched requests according to the image stored in the Mask Register and selects the highest priority non-masked interrupt request. When the ICU selects an interrupt request, it informs the Computer Control Unit. Then, in response to the interrupt processing microroutine, it outputs the Linkage Pointer corresponding to the selected interrupt.

This chapter describes the Interrupt Control Unit's design and its response to specific microinstructions. This description is presented in three sections. The first section describes the internal architecture of the AM 2914 Priority Interrupt Encoder which is the main component of the ICU. The second section presents the ICU design. The final section details the ICU's response to specific microinstructions.

AM 2914 Block Diagram Description

The internal block diagram of the AM 2914, shown on Figure 18, is described in this section to provide background for later discussion. The key elements of the AM 2914 block diagram are the Microinstruction Decoder, Interrupt Latch, Interrupt Register, Mask Register, Status Register, Interrupt Enable Flip Flop, and Interrupt Request Logic. These components are described below.

The AM 2914 performs sixteen different operations under the control of a 4-bit microinstruction field and the $\overline{IE}$ (Interrupt Instruction Enable)

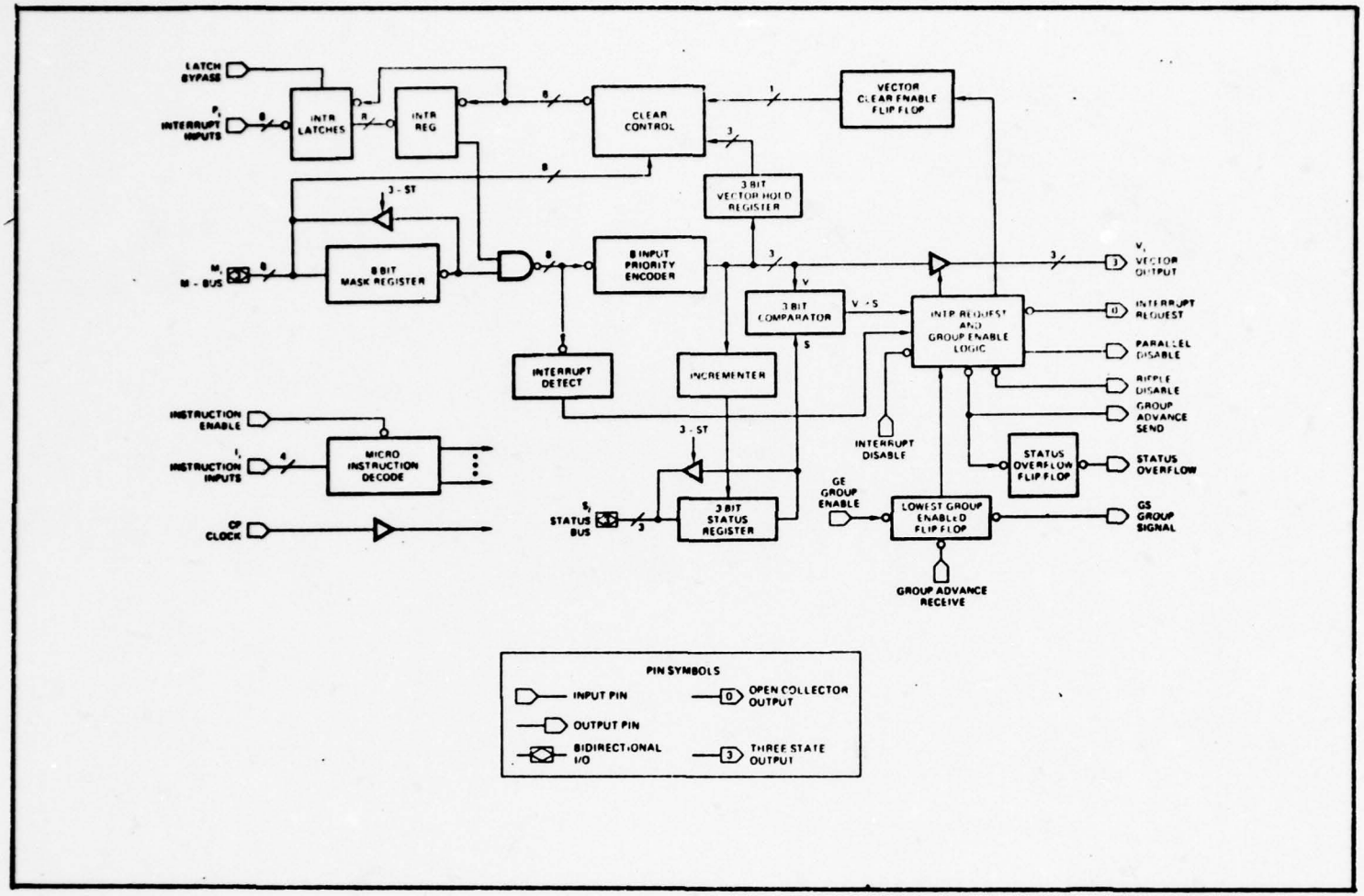


Fig. 18. AM2914 Block Diagram
(From Ref 5:3)

signal. These sixteen operations, listed in Table B15, are discussed later in this chapter. The AM 2914's Microinstruction Decoder will only decode this microinstruction field when the $\overline{IE}$ signal is "low". Otherwise, the microinstruction field is ignored.

The AM 2914 receives interrupt requests on eight input lines (a "low" level is a request). When the latch bypass signal is "low", the 8-bit Interrupt Latch catches negative pulses on the input lines. When the bypass signal is "high", the latch is bypassed and the interrupt inputs are clocked into the Interrupt Register.

The Mask Register holds eight mask bits which are used to mask (inhibit) individual interrupt inputs. Masking is accomplished by logically AND'ing the inverted outputs of the Mask Register with the corresponding outputs of the Interrupt Register. The 8-bit result, which represents all non-masked interrupt requests, is routed to the Interrupt Detector and the Priority Encoder.

The Interrupt Detector is an eight input OR gate which inspects the non-masked interrupt requests and sends a signal to the Interrupt Request Logic if one or more non-masked requests are detected.

The Priority Encoder also examines the non-masked interrupt requests. If one or more requests are present, the Priority Encoder determines which request has the highest priority and then identifies that request with a 3-bit binary interrupt vector. When the Vector Read microinstruction (described later) is executed, the vector is output over three-state lines and it is also latched into the Vector Hold Register. This stored vector can be used, at a later time, to clear the interrupt request from the Interrupt Latch.

The 3-bit Status Register points to the lowest priority at which an

interrupt will be accepted. The contents of this register are compared with the interrupt vector and an acceptance signal is only sent to the Interrupt Request Logic when the vector is greater than or equal to the status value. During the Vector Read microinstruction, the Status Register is automatically loaded with the value of the interrupt vector plus one. Automatically loading the Status Register in this manner prevents multiple requests from the same interrupt input.

The Interrupt Request Logic determines if a valid interrupt request is present at the interrupt inputs. To make this determination, it checks the output of the Interrupt Detector to verify that a non-masked request is present and it verifies that the corresponding vector is greater than or equal to the Status Register value. If both conditions are met and if the Interrupt Enable Flip Flop is set, the Interrupt Request Logic signals that a valid interrupt request is present by pulling the open collector Interrupt Request Flag to a "low" level.

The Lowest Group Enable Flip Flop, Group Enable Logic, and associated control lines (i.e. Parallel Disable, Ripple Disable, Group Advance Send, Status Overflow, Group Send, Group Advance Receive, and Group Enable) are used to cascade several AM 2914 devices (Ref. 5:15-16).

Interrupt Control Unit Design

The internal block diagram of the AM 2914 was described in the previous section of this chapter. This section presents the Interrupt Control Unit (ICU) design.

The basic design of the Interrupt Control Unit, shown on Figure 19, was adapted from the AM 2914's applications literature (Ref 5:35-39). In this

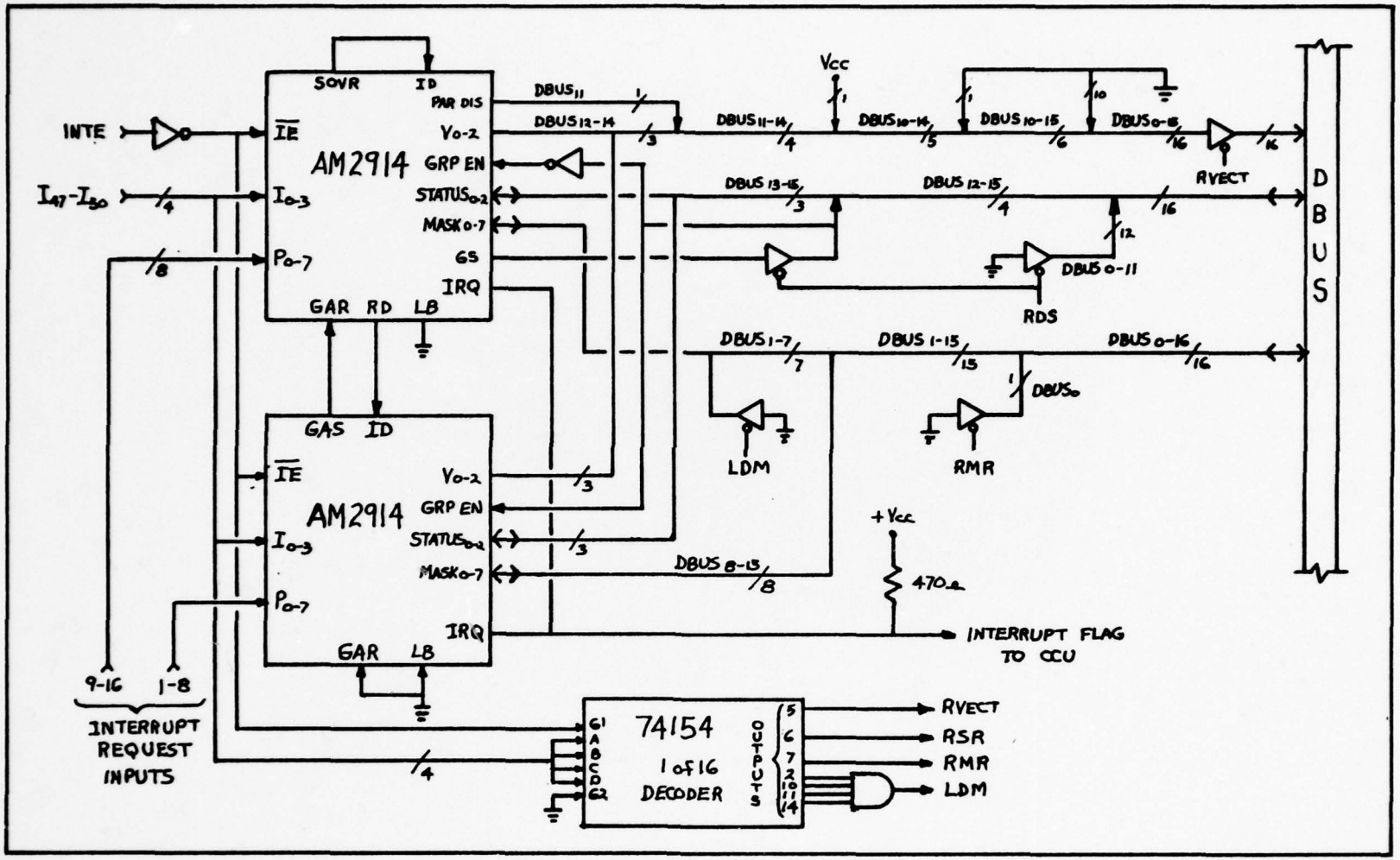


Fig. 19. Interrupt Control Unit
Block Diagram

design, two AM 2914 devices are cascaded so that they operate as a sixteen level interrupt system.

The Interrupt Latches of the two AM 2914's function as the 16-bit Interrupt Pending Register which was described in Chapter II. The sixteen interrupt request lines of the interrupt system are connected directly to the AM 2914 interrupt inputs. The Interrupt Latches are enabled by tying the Latch Bypass signal "low", so that the latch will catch both pulses and levels on the interrupt request inputs. Once an interrupt request is captured, it remains latched until the interrupt is recognized or until the latch (Interrupt Pending Register) is cleared.

The Mask Registers of the two AM 2914 devices function as the 16-bit Mask Register defined in Chapter II. The three-state input/output lines of the Mask Register are interfaced directly with the processor's D-BUS. To insure that Interrupt 16 (power down) is not masked, three-state buffers force the most significant bit of the Mask Register "low" during mask read/write operations.

The interrupt vector output lines of the two AM 2914's are cascaded to form a 4-bit interrupt vector. Hardware interconnections map this vector into the 16-bit Linkage Pointer address (defined in Chapter II; Table III). Three-state buffers enable the Linkage Pointer address onto the D-BUS during the Vector Read operation.

The operations of the two AM 2914 Status Registers are cascaded so that these registers function as a single 4-bit Status Register. The Status Register input/output lines are interfaced with the lower 4-bits of the D-BUS, using three-state buffers, so that the status value can be loaded or read from the D-BUS.

The open collecter Interrupt Request Flags of the two AM 2914 devices

are tied together and this flag is routed to the Condition Test Multiplexer in the Computer Control Unit. The processor's microprogram will test this flag at the end of each machine instruction to determine if a valid interrupt request is present.

ICU Microinstructions

In the preceding sections of this chapter, the AM 2914 internal block diagram was described and the Interrupt Control Unit's design was presented. In this section, the microinstructions which control the ICU will be described.

Microinstruction Descriptions. The ICU executes sixteen microinstructions under the control of the 4-bit microinstruction field (I47 - I50), shown on Table B15. These microinstructions are only executed if the Interrupt Instruction Enable (INTE) microcommand is issued by the microinstruction Command field (Table B9). The sixteen microinstructions are described below.

(1) Master Clear. The Master Clear (IMCL) microinstruction clears the Interrupt Pending Register, Mask Register, and Status Register. This instruction also sets the Interrupt Enable Flip Flop so that all interrupts are enabled.

(2) Clear All Interrupts. The Interrupt Pending Register is cleared by the Clear All Interrupts (CAI) microinstruction.

(3) Clear Interrupts from BUS. The Clear All Interrupts from BUS (MBC) microinstruction clears the Interrupt Pending Register bits which have corresponding D-BUS bits set equal to one.

(4) Clear Interrupts from Mask Register. The Interrupt Pending Register bits which have their corresponding Mask Register bits set equal

to one are cleared by the Clear Interrupt from Mask Register (IMC) microinstruction.

(5) Clear Interrupts, Last Vector Read. The Clear Interrupts, Last Vector Read (CLV) microinstruction clears the Interrupt Pending Register bit which is associated with the last interrupt vector read.

(6) Read Vector. The Read Vector (RDV) microinstruction is used to read the Linkage Pointer address which is associated with the highest priority non-masked interrupt request. This instruction also loads the value of the interrupt vector plus one into the Status Register. In addition, it sets the Vector Clear Enable Flip Flop and loads the current vector value into the Vector Hold Register so that it can be used later by the Clear Interrupt, Last Vector Read instruction.

(7) Read Status Register. During the Read Status Register (RSR) microinstruction, the contents of the Status Register are output onto the D-BUS.

(8) Read Mask Register. The Mask Register outputs are enabled onto the D-BUS when the Read Mask Register (RMR) microinstruction is executed.

(9) Set Mask Register. The Set Mask Register (SMR) microinstruction sets all bits in the Mask Register to one. This inhibits recognition of all interrupt requests.

(10) Load Status Register. The Load Status Register (LSR) microinstruction loads the data from the lower 4-bits of the D-BUS into the Status Register.

(11) Bit Clear Mask Register. The Bit Clear Mask Register (BCM) microinstruction clears those Mask Register bits which have corresponding D-BUS bits equal to "one". Other Mask Register bits are not affected.

(12) Bit Set Mask Register. The Bit Set Mask Register (BSM) microinstruction sets those Mask Register bits which have corresponding D-BUS bits equal to one. Other Mask Registers are not affected.

(13) Clear Mask Register. The entire Mask Register is cleared by the Clear Mask Register (CMR) microinstruction. This enables all interrupts subject to the state of the Interrupt Enable Flip Flop and the Status Register.

(14) Disable Interrupt Request. All interrupt requests are disabled by the Disable Interrupt Request (DIR) microinstruction.

(15) Load Mask Register. The Load Mask Register (LMR) microinstruction loads data from the D-BUS into the Mask Register.

(16) Enable Interrupt Requests. The Enable Interrupt Requests (EIR) microinstruction enables all interrupt requests subject to the contents of the Mask Register and the Status Register.

Microinstruction Comments. The interrupt system requirements, presented in Chapter II, specify that Interrupts 14, 15, and 16 cannot be disabled by the Disable Interrupt (DSBL) macroinstruction. This requirement restricts the use of the Disable Interrupt Request (DIR) microinstruction since the DIR microinstruction disables all interrupts, including Interrupts 14, 15, and 16. To overcome this problem, the Status Register will be used to disable interrupts. When the Disable Interrupt (DSBL) macroinstruction is executed, the Status Register will be loaded with the value 14_{10} using the Load Status Register (LSR) microinstruction. This status value will inhibit recognition of all interrupts except Interrupts 14, 15, and 16. Later, when the Enable Interrupt (ENBL) macroinstruction is executed, the Status Register will be cleared to enable all interrupts, subject to the contents of the Mask Register.

When an interrupt is recognized, the interrupt system's requirements specify that all interrupts will be automatically disabled (except Interrupt 16). This requirement allows the macro level interrupt handling routine an opportunity to direct the interrupting device to remove its interrupt request and to perform interrupt nesting actions before the next interrupt is recognized. To satisfy this requirement, the Status Register will be automatically loaded with a value 16_{10} whenever an interrupt is recognized.

Summary

The Interrupt Control Unit's (ICU's) design was presented in this chapter along with a description of the ICU's response to specific microcontrol signals. In addition, this chapter discussed micro-programming restrictions which are necessary to satisfy the interrupt system requirements.

The next chapter completes the description of the processor's hardware by presenting the design of the Input/Output Interface Unit (IOIU).

VII. Input/Output Interface Unit Design

The Input/Output Interface Unit (IOIU) interfaces the processor with the microcomputer's I-BUS. It also interfaces the processor with the Discrete Input/Output Channel. In addition, the IOIU contains the interval timers (Timer A and Timer B) and the processor control panel.

The IOIU block diagram, Figure 20, shows that the IOIU design is divided into four subunits: (1) I-BUS Interface; (2) Discrete Input/Output Interface; (3) Interval Timers; and (4) Processor Control Panel. These subunits will be discussed in separate sections below.

I-BUS

The microcomputer I-BUS, described in Chapter II, transfers information between components of the microcomputer (i.e. Processor, Memory Modules, PIO Channels, and DMA Channel). Before the processor can transfer information over the I-BUS, it must first access the bus; then it must execute the appropriate transfer sequence (Master to Slave Send cycle or Master from Slave Receive cycle).

The I-BUS Interface Unit (Figure 21) is an autonomous transfer controller which performs the I-BUS access and transfer control functions for the processor. In addition, the I-BUS Interface Unit takes action to interleave the I-BUS operations of the processor and the DMA Channel. The I-BUS transfer functions and DMA Control functions of the I-BUS Interface Unit are described below.

I-BUS Transfer Function. The main function of the I-BUS Interface Unit is to perform the I-BUS access and transfer control operations for the processor. The I-BUS Interface unit responds to the PRRQ (PIO Read

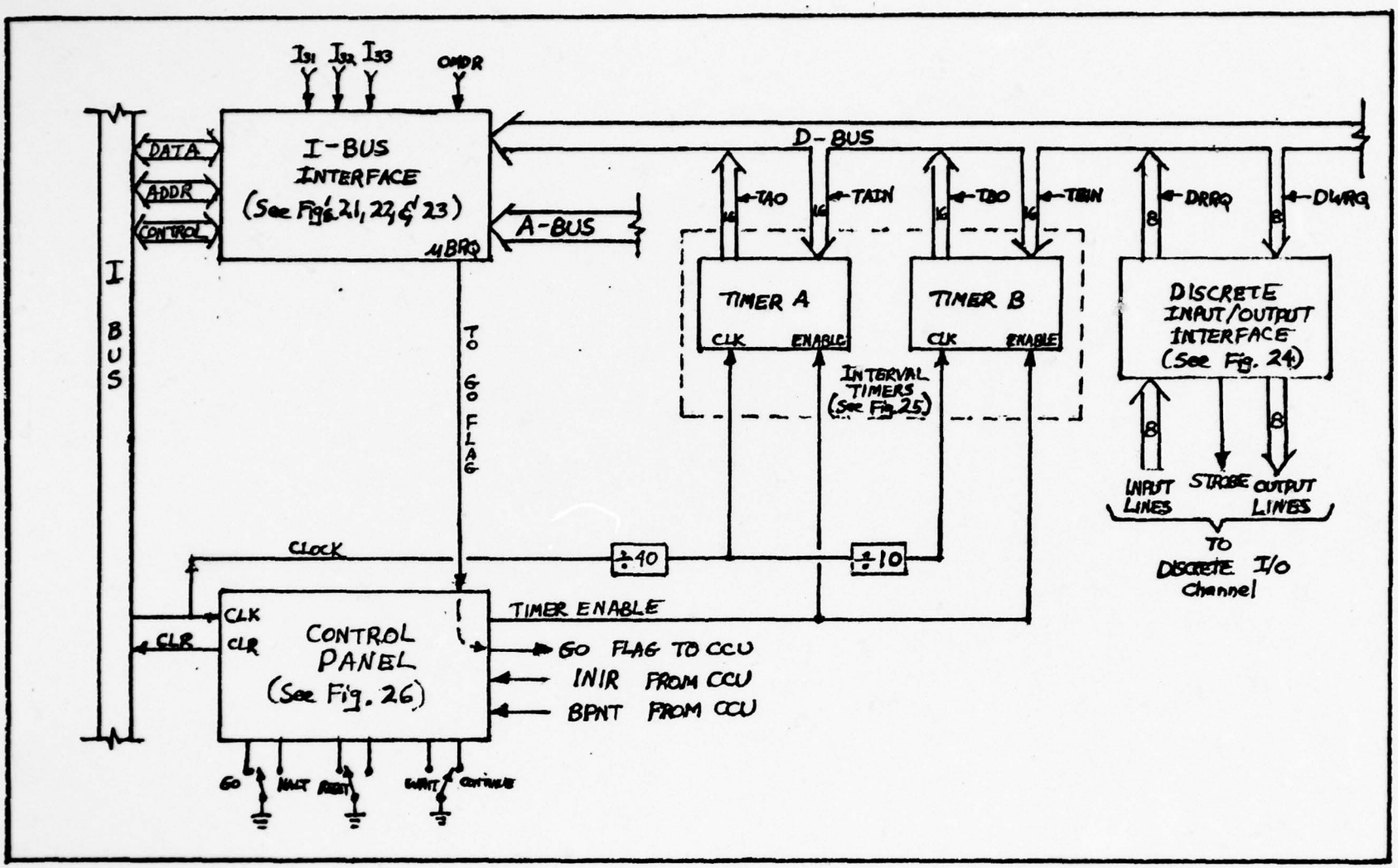


Fig. 20. Input/Output Interface Unit Block Diagram

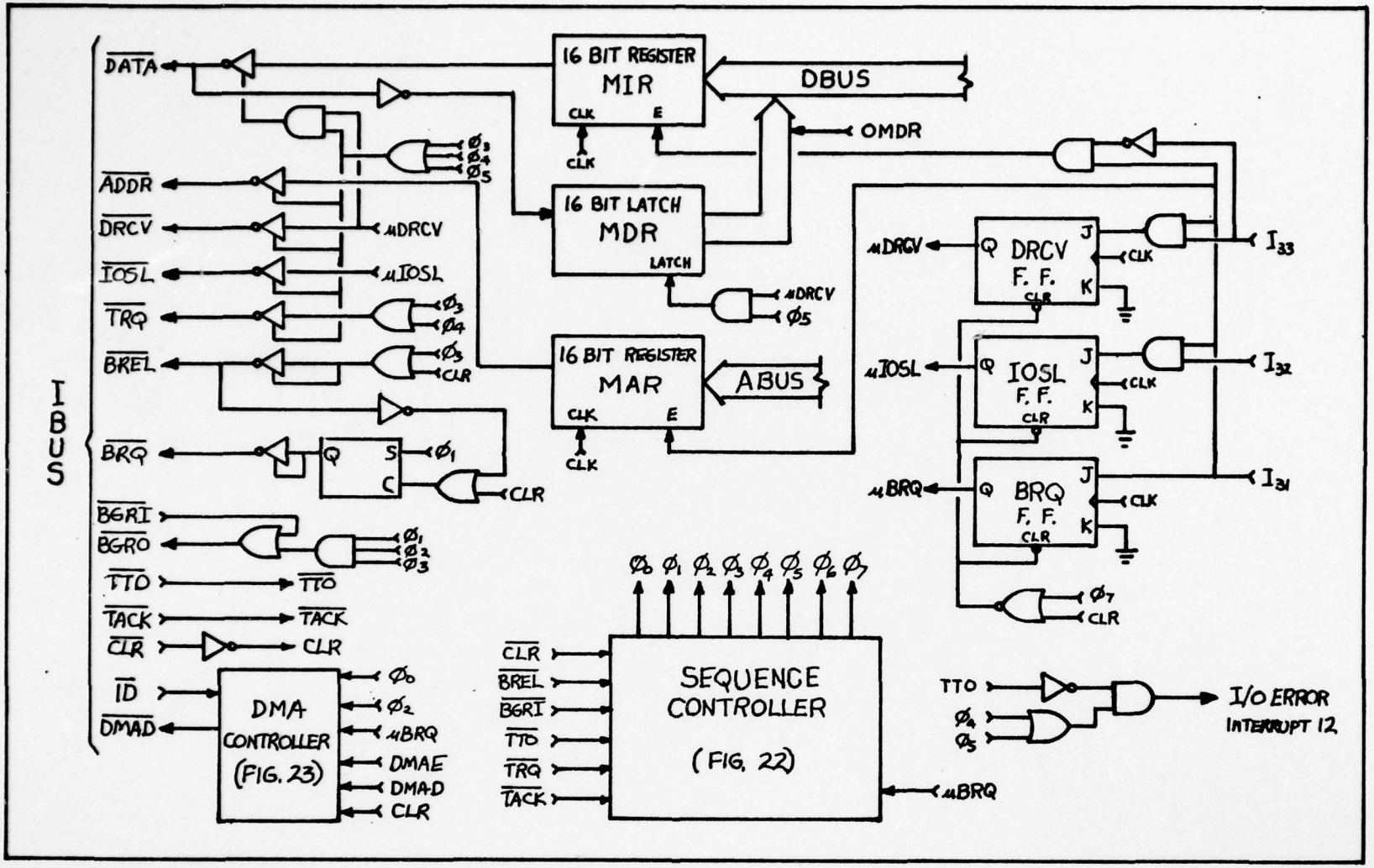
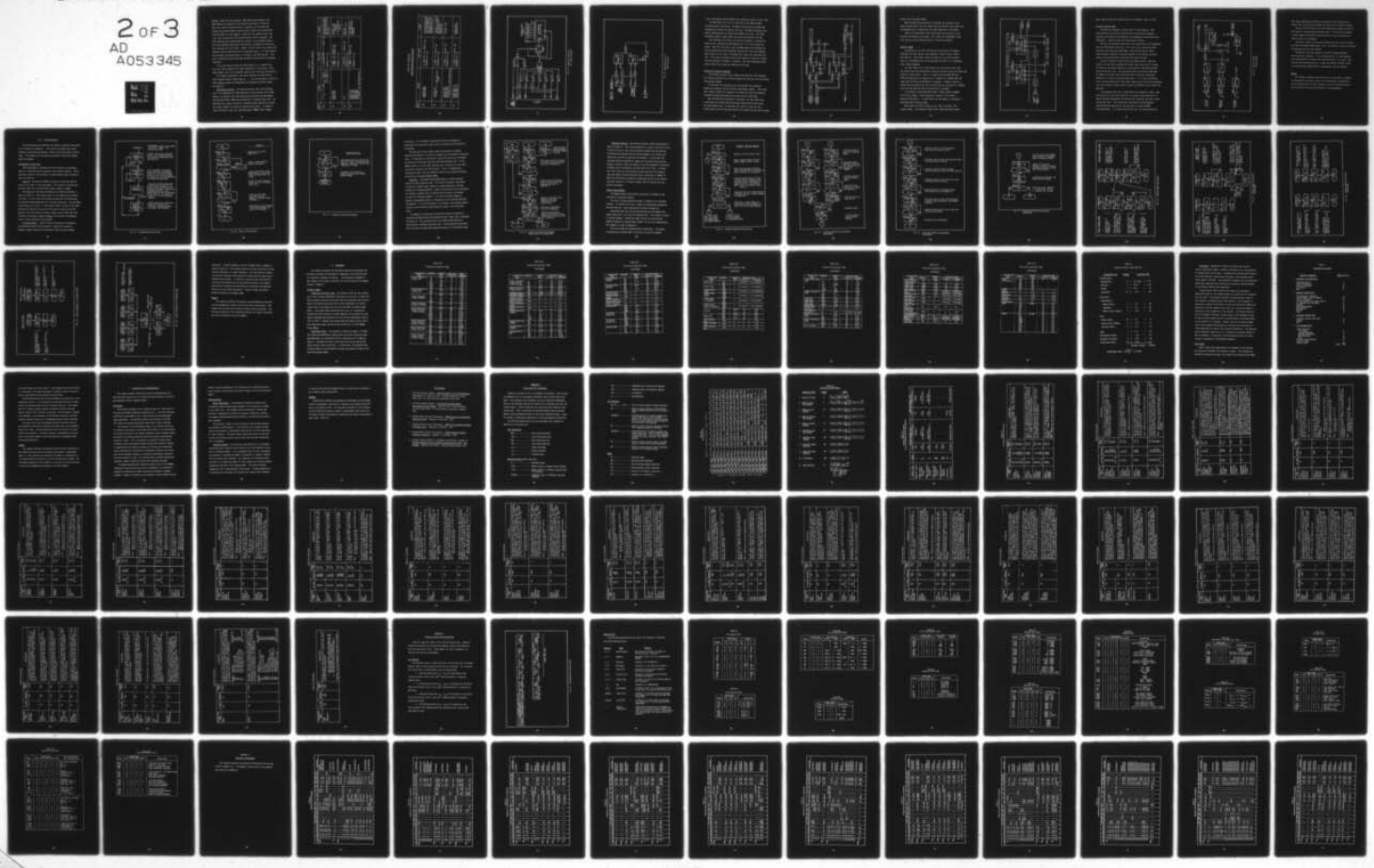


Fig. 21. I-BUS Interface for IOIU

AD-A053 345 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
DESIGN OF THE PROCESSOR FOR SOFTWARE COMPATIBLE AVIONIC COMPUTE--ETC(U)
DEC 77 F G THOUROT

UNCLASSIFIED AFIT/GCS/EE/77-10 NL

2 OF 3
AD
A053345



Request), PWRQ (PIO Write Request), MRRQ (Memory Read Request), and MWRQ (Memory Write Request) microinstructions defined in Table B10. When one of these microinstructions is executed, the Interface Unit latches the transfer address from the A-BUS, latches the transfer data from the D-BUS (if write request), forces the "GO" condition test flag "low", and initiates the I-BUS access/transfer control sequences. The I-BUS interface then accomplishes the transfer in an autonomous manner. The I-BUS interface signals when the transfer is complete by raising the "GO" condition test flag "high". (NOTE: The "GO" flag is also used by the processor control panel which will be described later.) If a read request was performed, the received data is available in the MDR register. This data is enabled onto the D-BUS when the OMDR microinstruction is executed (Table B7).

If an error occurs during the I-BUS transfer (i.e. Transfer Time Out), the I-BUS interface will signal completion of the transfer in the normal manner, but it will generate a parity error interrupt (Interrupt 8).

The internal operations of the I-BUS Interface are controlled by eight sequence control signals ($\emptyset_0$, $\emptyset_1$, ... , $\emptyset_7$) as described on Table VII. These sequence signals are generated by the sequence controller shown on Figure 22.

DMA Control Function. The second function of the I-BUS Interface unit is to interleave the I-BUS operations of the processor and the DMA Channel. In addition, the I-BUS Interface must disable the DMA Channel when the DMA Disable (DMAD) macroinstruction is executed (Table A4). The DMA Controller, shown on Figure 23, implements these operations with two flip flops (Disable flip flop and Interleave flip flop). If either of these flip flops is set, the I-BUS DMA Disable ($\overline{DMAD}$) signal becomes

Table VII

I-BUS Interface Control Sequence

STATE	ACTION	NEXT STATE	COMMENT
$\emptyset_0$	IF (I_{31} = 1); THEN I_{31} → uBRQ f.f. I_{32} → uIOSL f.f. I_{33} → uDRCV f.f.	IF (uBRQ·$\overline{BGRI}$·$\overline{BREL}$ = 1): THEN $\emptyset = \emptyset_1$ OTHERWISE: $\emptyset = \emptyset_0$	INITIAL STATE, WAIT FOR MICROINSTRUCTION THEN WAIT FOR I-BUS AVAILABLE

$\emptyset_1$	BLOCK BGRO
	ASSERT BRQ
		IF ($\overline{BREL}$ = 0); THEN $\emptyset = \emptyset_0$
		IF ($\overline{BGRI}$ = 0); THEN $\emptyset = \emptyset_2$
		OTHERWISE; $\emptyset = \emptyset_1$
			REQUEST I-BUS
			IF SUCCESSFUL THEN $\emptyset_2$
			IF NOT SUCCESSFUL THEN $\emptyset_0$

$\emptyset_2$	WAIT	IF ($\overline{TRQ}$·$\overline{TACK}$ = 1); THEN $\emptyset = \emptyset_3$ OTHERWISE: $\emptyset = \emptyset_2$	WAIT FOR QUIESENT I-BUS
$\emptyset_3$	ASSERT TRQ ASSERT DRCV ASSERT IOSL ASSERT BREL ASSERT ADDR IF (uDRCV = 0); THEN ASSERT DATA	IF ($\overline{BGRI}$ = 1); THEN $\emptyset = \emptyset_4$ OTHERWISE: $\emptyset = \emptyset_3$	TAKE CONTROL OF I-BUS AND INITATE TRANSFER

Table VII (continued)

I-BUS Interface Control Sequence

STATE	ACTION	NEXT STATE	COMMENT
$\emptyset_4$	RELEASE BREL UNBLOCK BGRO	IF (TACK·TTO = 0); THEN $\emptyset = \emptyset_5$ OTHERWISE; $\emptyset = \emptyset_4$	ALLOW RESOLUTION OF NEXT BUS MASTER AND WAIT FOR TRANSFER ACKNOWLEDGE OR TIME TRANSFER TIME OUT
$\emptyset_5$	RELEASE TRQ IF (uDRCV = 1); THEN LATCH DATA IN MDR	IF ($\overline{\text{TRQ}}$ = 1); THEN $\emptyset = \emptyset_6$ OTHERWISE; $\emptyset = \emptyset_5$	TRANSFER IS COMPLETE BUT MUST WAIT FOR SLOW DEVICE TO RELEASE TRQ
$\emptyset_6$	RELEASE DRCV RELEASE IOSL RELEASE ADDR IF (uDRCV = 0); THEN RELEASE DATA	IF ($\overline{\text{TACK}}$ = 1); THEN $\emptyset = \emptyset_7$ OTHERWISE; $\emptyset = \emptyset_6$	RELEASE I-BUS
$\emptyset_7$	CLEAR uBRQ f.f. CLEAR uIOSL f.f. CLEAR uDRCV f.f.	IF (uBRQ = 0); THEN $\emptyset = \emptyset_0$ OTHERWISE; $\emptyset = \emptyset_7$	CLEAR I-BUS INTERFACE UNIT

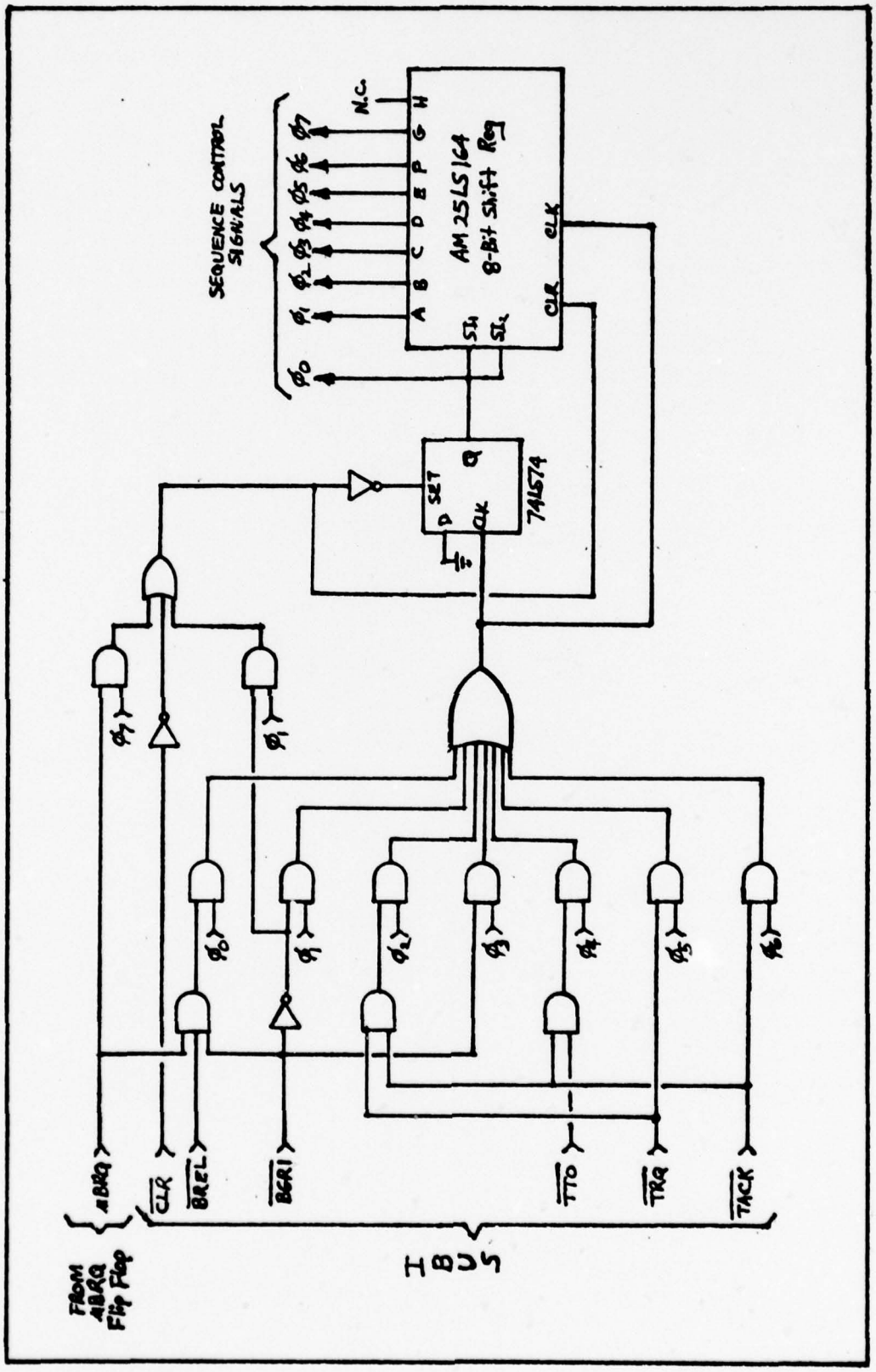


Fig. 22. Sequence Controller for I-BUS Interface

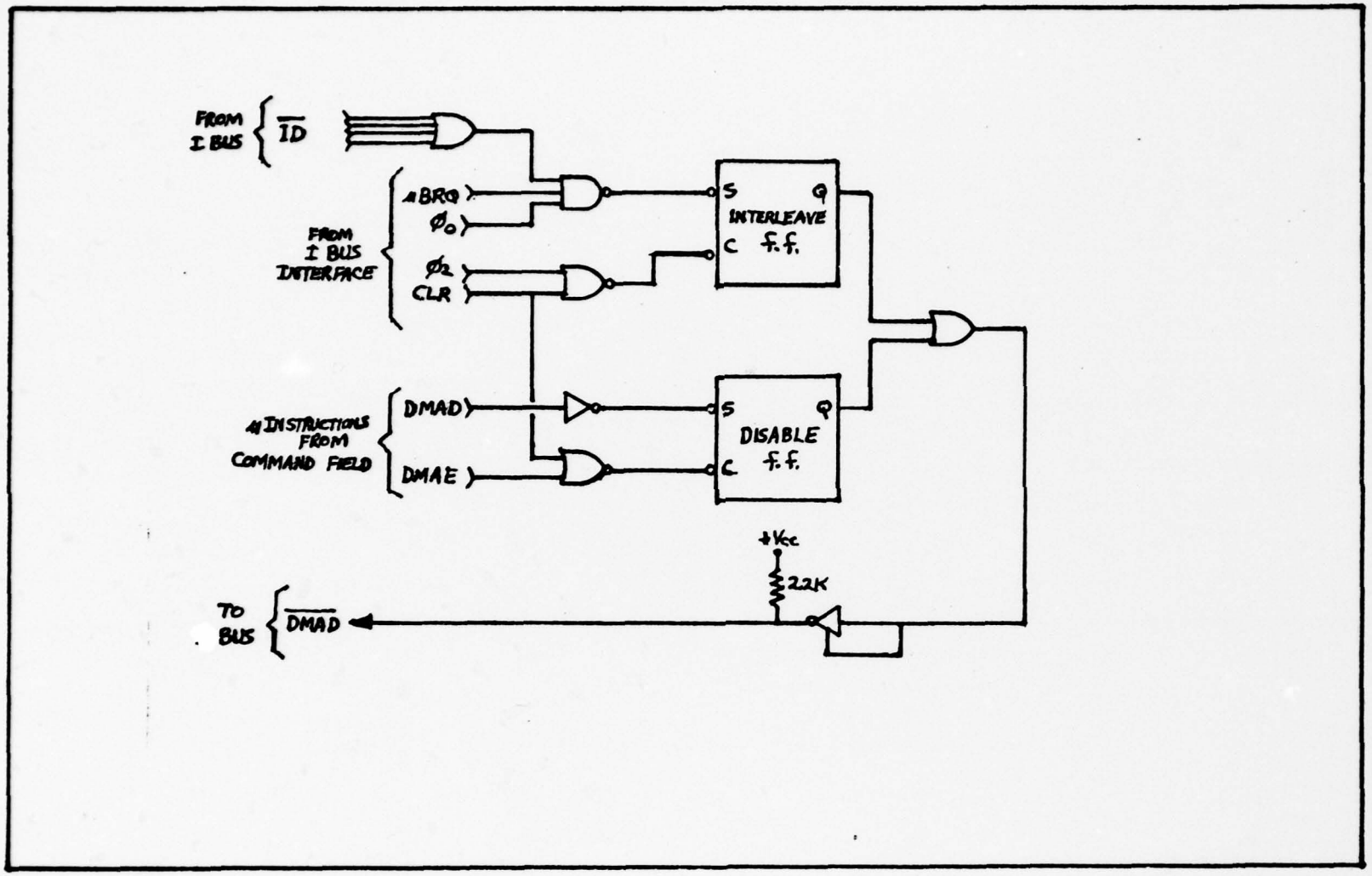


Fig. 23. DMA Controller for I-BUS Interface

"true" and prevents the DMA Channel from requesting control of the I-BUS.

The DMA Disable flip flop is controlled by the DMAD and DMAE microinstructions (Table B9). The DMAD microinstruction disables DMA operations by setting the Disable flip flop. The DMAE microinstruction enables DMA operations by clearing the Disable flip flop. The I-BUS Clear/Reset Control signal (CLR) also clears the Disable flip flop.

The Interleave flip flop is set when the processor is waiting for access to the I-BUS and the DMA Channel (ID = 15) is the current bus master. When this flip flop is set, the DMA Channel will complete its current transfer and then relinquish I-BUS control When the processor gains control of the I-BUS, it automatically clears the Interleave flip flop. This allows the higher priority DMA Channel to regain bus control once the processor's transfer is complete. The I-BUS Clear/Reset (CLR) control signal also clears the Interleave flip flop.

Discrete I/O Channel Interface

The previous section of this chapter described the I-BUS interface. This section describes the interface between the processor and the Discrete Input/Output Channel.

The Discrete I/O Interface of the IOIU buffers data transfers between the processor and the Discrete Input/Output Channel. This interface, shown on Figure 24, responds to the DWRQ (Discrete Write Request) and DRRQ (Discrete Read Request) microinstructions (Table B10).

When the DWRQ microinstruction is executed, the lower eight bits of the D-BUS are latched into the discrete output buffer which drives the eight output lines. At the same time a pulse is sent over the strobe output line to inform the Discrete I/O Channel that new data has been

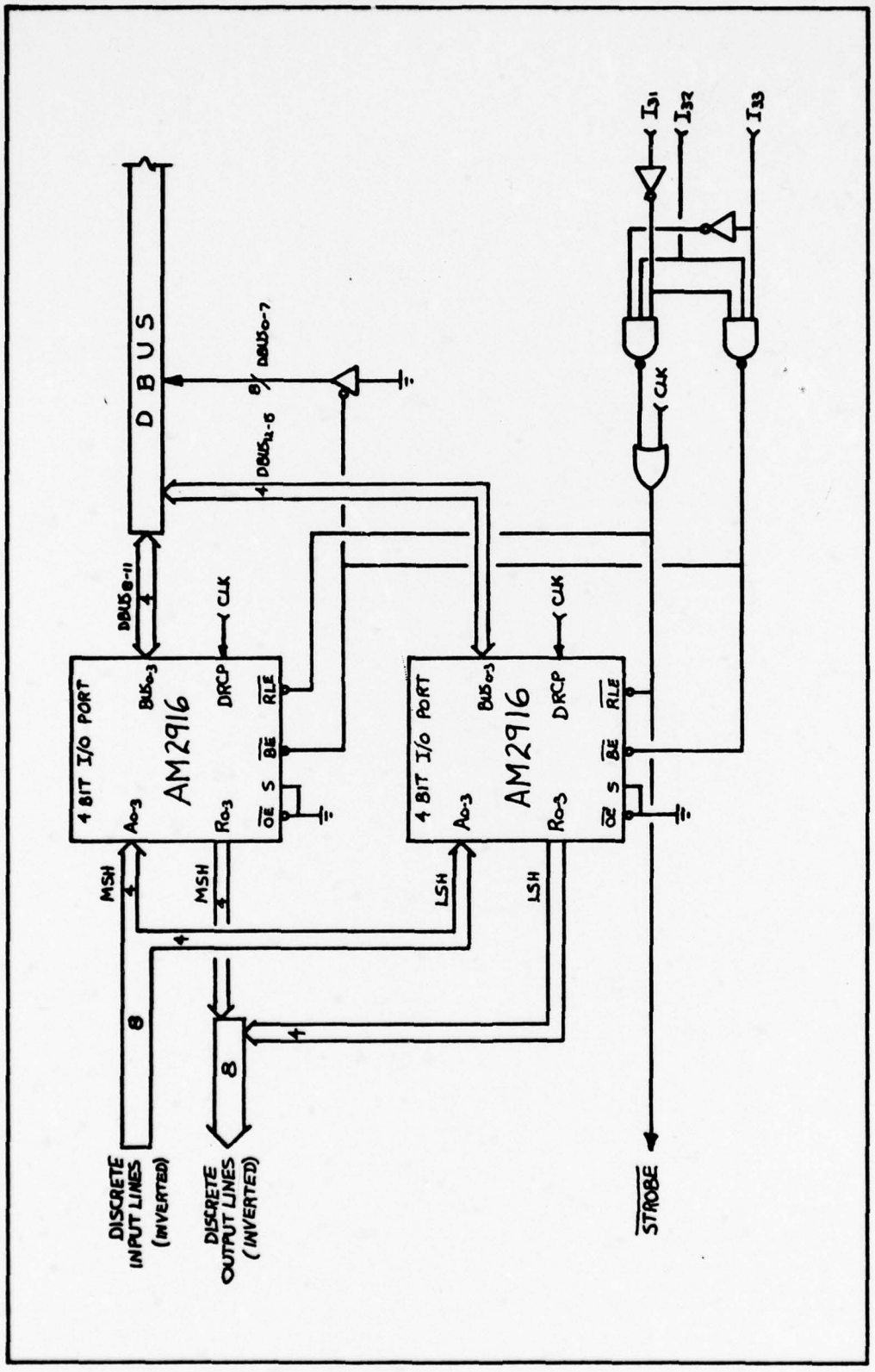


Fig. 24, Discrete Data Channel Interface

written into the output buffer.

When the DRRQ microinstruction is executed, the contents of the eight discrete input lines are clocked into the discrete input buffer and the buffered data is enabled onto the lower eight bits of the D-BUS.

Signals on input/output lines which connect the interface to the Discrete I/O Channel are inverted. This allows the input/output lines to be driven by either open-collector or three state buffers.

Interval Timers

In addition to the I-BUS interface and the Discrete I/O Channel Interface, the IOIU contains the processor's interval timers (Timer A and Timer B). These timers, shown on Figure 25, are 16-bit counters. Timer A is incremented every 10 microseconds and Timer B is incremented every 100 microseconds.

The TAIN (Timer A In), TBIN (Timer B In), TAO (Timer A Out), and TBO (Timer B Out) microinstructions (Tables B7 and B9) control the inputs and outputs of these timers. Timer A is loaded from the D-BUS when the TAIN microinstruction is executed; and it enables its contents onto the D-BUS when the TAO microinstruction is executed. Likewise, Timer B is loaded from the D-BUS by the TBIN microinstruction; and enables it contents onto the D-BUS when the TBO microinstruction is executed.

An interrupt is generated when either of these timers increment from $FFFF_H$ to 0000_H. Timer A generates Interrupt 10 and Timer B generates Interrupt 8. If these timers are not loaded, an interrupt is generated after $65{,}536_{10}$ counts.

Both timers are reset to 0000_H by the I-BUS Clear/Reset (CLR) control signal. The Processor Control Panel (described below) stops

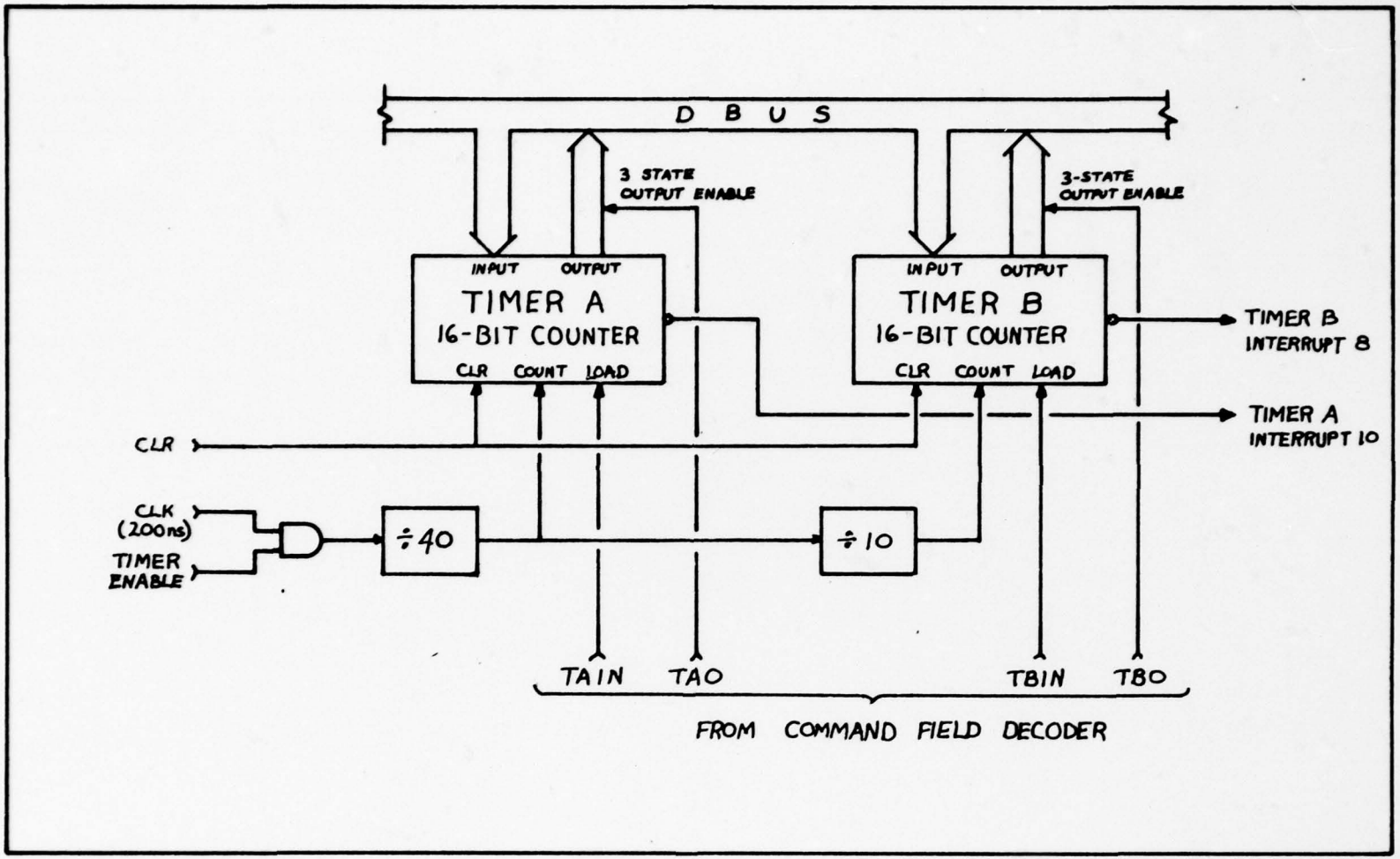


Fig. 25. Interval Timers
Block Diagram

these timers during test functions such as a breakpoint, wait, or halt.

Processor Control Panel

The IOIU also includes a control panel for the processor. This panel performs basic control functions such as halting the processor, resetting the processor (Clear/Reset), and responding to macroinstruction breakpoints. Figure 26 shows the control panel's design.

The halt and breakpoint functions of the control panel are implemented with the "GO" condition test flag. This flag is also used by the I-BUS Interface Unit (previously described). Using the GO flag for these three different functions is possible because of the mutually exclusive nature of the microroutines associated with these functions.

The halt function is controlled by the GO/HALT switch. When the switch is at the "HALT" position the control panel waits until it senses the INIR (In Instruction Register) microinstruction (Table B8) and then pulls the GO flag "low". The instruction fetch microroutine (described in Chapter VIII) tests the GO flag each time it loads the Instruction Register (i.e. start of new macroinstruction). If it finds the GO flag "low", the microroutine will halt (loop, repeating the INIR microinstruction) until the GO flag is forced "High" by moving the GO/HALT switch to the "GO" position.

The breakpoint function is controlled by the Breakpoint switch. When that switch is in the "WAIT" position, the control panel waits until it senses the BPNT (Breakpoint) microinstruction (Table B9) and then it pulls the GO flag "low". The microroutine associated with the Breakpoint macroinstruction tests the GO flag each time it issues the BPNT microinstruction. If it finds the GO flag "low", the microroutine will

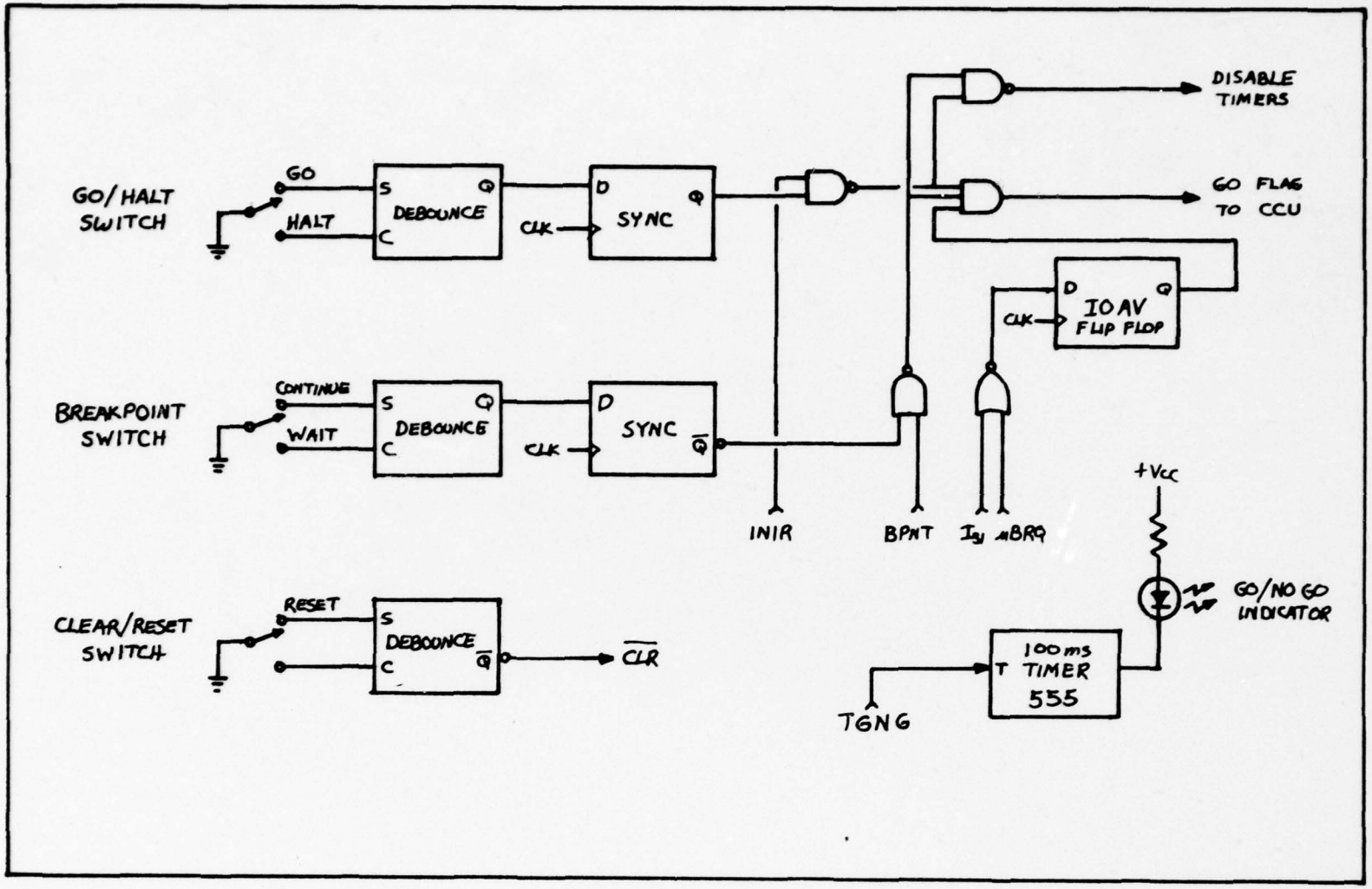


Fig. 26. Control Panel
Block Diagram

halt (loop, repeating the BPNT microinstruction) until the GO flag is forced "high" by moving the Breakpoint switch to the "CONTINUE" position.

The use of the GO flag by the I-BUS Interface unit does not conflict with either of the operations described above. The instruction register is never loaded and the BPNT microinstruction is never issued while an I-BUS transfer is in progress.

The control panel's reset switch controls the flip flop which drives the I-BUS Clear/Reset ($\overline{CLR}$) control line. This switch is used to initialize the processor and all other devices.

The GO/HALT indicator responds to the TGNG (Trigger GO/NO GO) microinstruction (Table B9). This microinstruction is issued whenever the Trigger GO/NO GO Indicator (GO) macroinstruction is executed. Each time the TGNG microinstruction is issued, the GO/HALT indicator lights its Light Emitting Diode (LED) for approximately 100 milliseconds.

Summary

This chapter completed the description of the processor's hardware. It presented the functional design of the Input/Output Interface Unit (IOIU) and described the IOIU's response to specific microinstructions. The next chapter discusses the processor's microprogramming.

VIII. Microprogramming

The microprogram which emulates the software-capatible instruction set is presented in Appendix C. This chapter introduces that microprogram by describing the program's control flow and principal microroutines. This chapter also describes how different instruction address modes are decoded.

Microprogram Control Flow

The microprogram is organized into four principal microroutines: Power UP, Instruction Fetch, Execution, and Interrupt Handling. During execution, control of the processor is passed between these routines as shown on Figure 27.

Power Up. The Power UP (PWRU) microroutine initializes the processor at the start of the microprogram. This routine is entered automatically each time the Clear-Reset control signal is issued.

The sequence of operations performed by the PWRU microroutine is shown on Figure 28. This routine clears all ALU registers and control flip flops. It also clears the processor Status Word, Interrupt Mask, and Interrupt Pending Register; and it enables interrupts. Then the PWRU routine tests for a "Halt". If the GO/HALT switch is moved to the "HALT" position the routine will wait until the switch is moved to the "GO" position. Then the routine initiates a memory read to fetch the first instruction from memory (address 0000_{HEX}) and transfers microprogram control to the Instruction Fetch Routine.

Instruction Fetch. Figure 29 shows the operations performed by the Instruction Fetch (IF) microroutine. Before this routine is entered, a memory read must be initiated to fetch the next machine

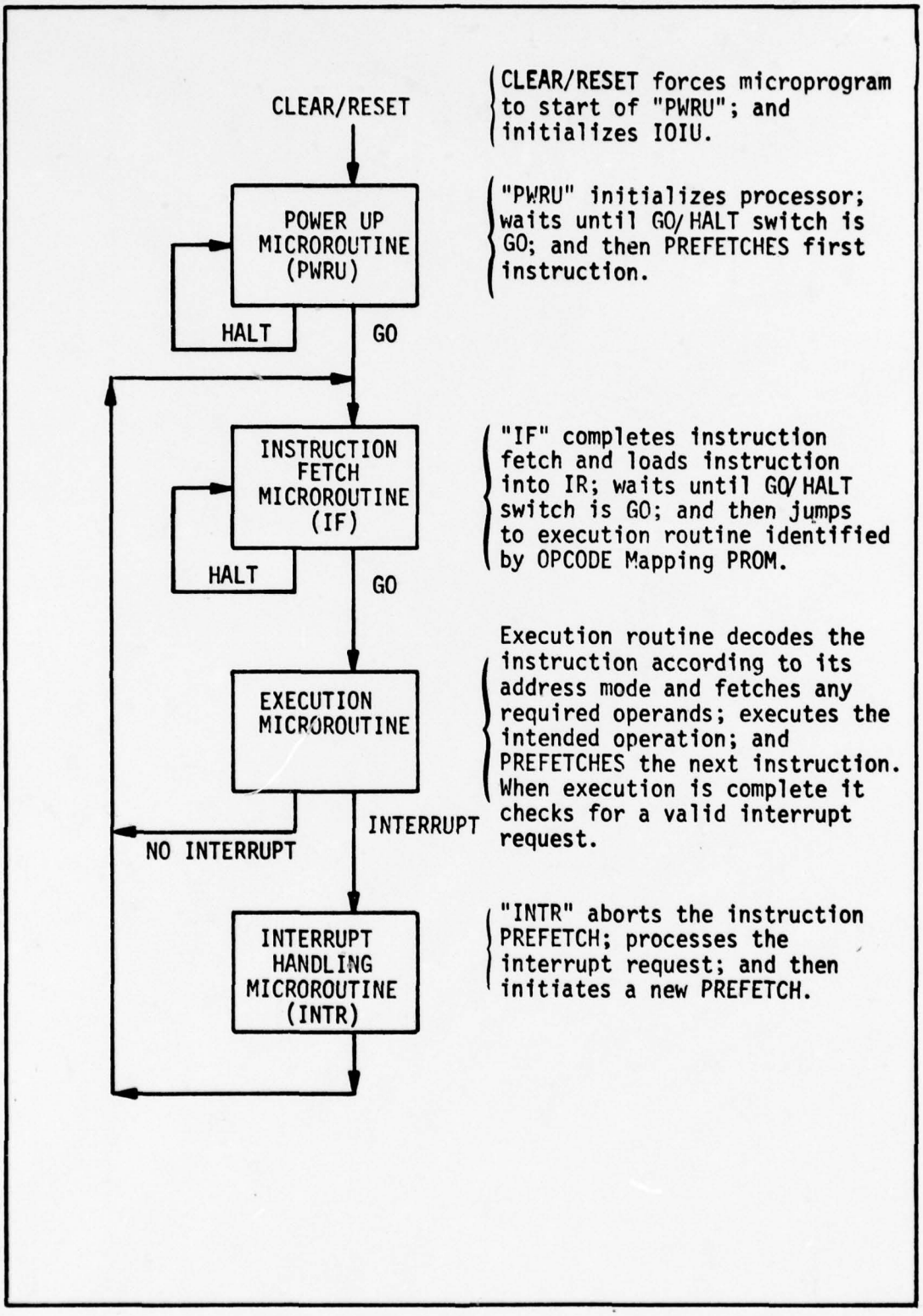


Fig. 27. Microprogram Control Flow

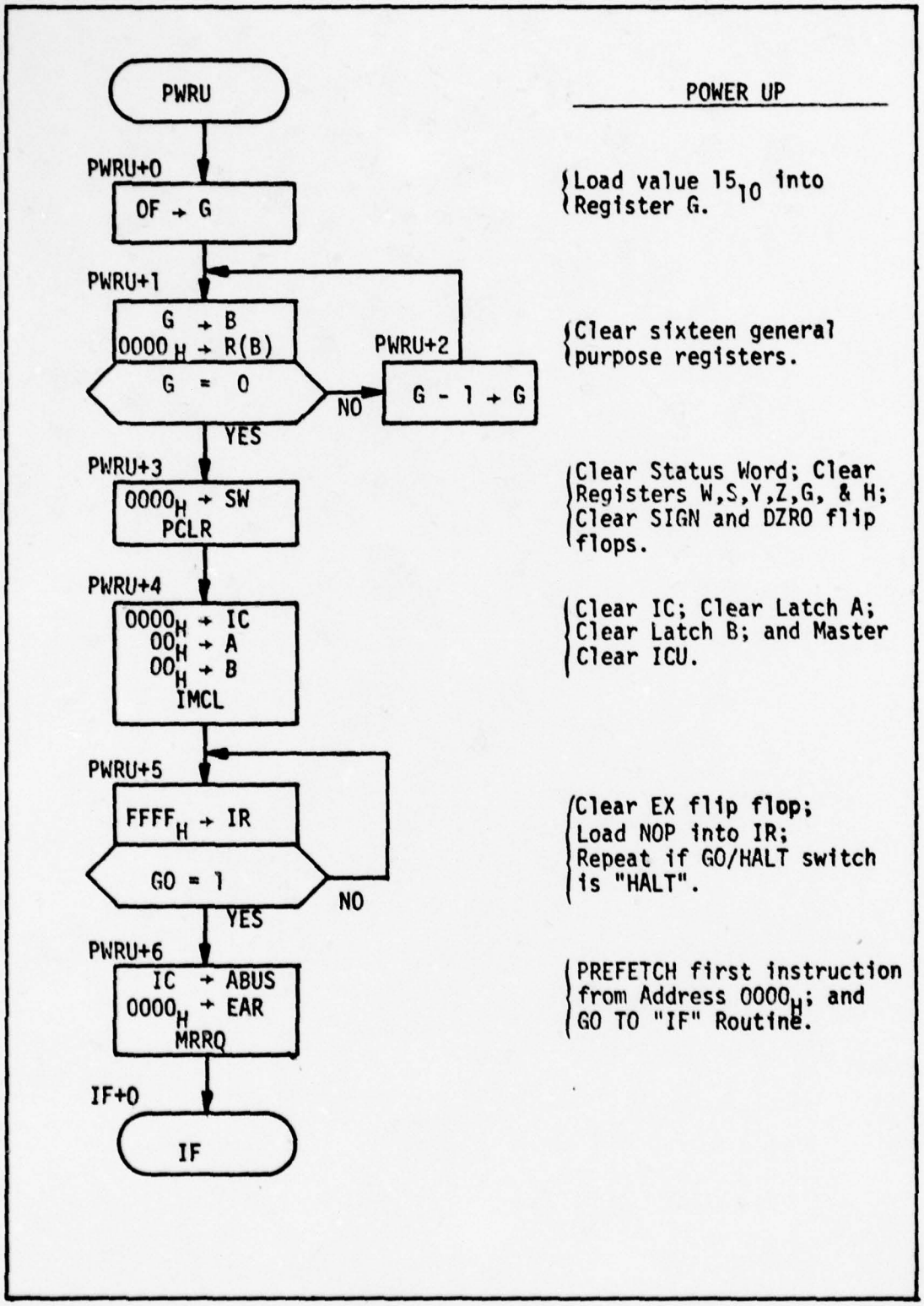


Fig. 28. Power Up Microroutine

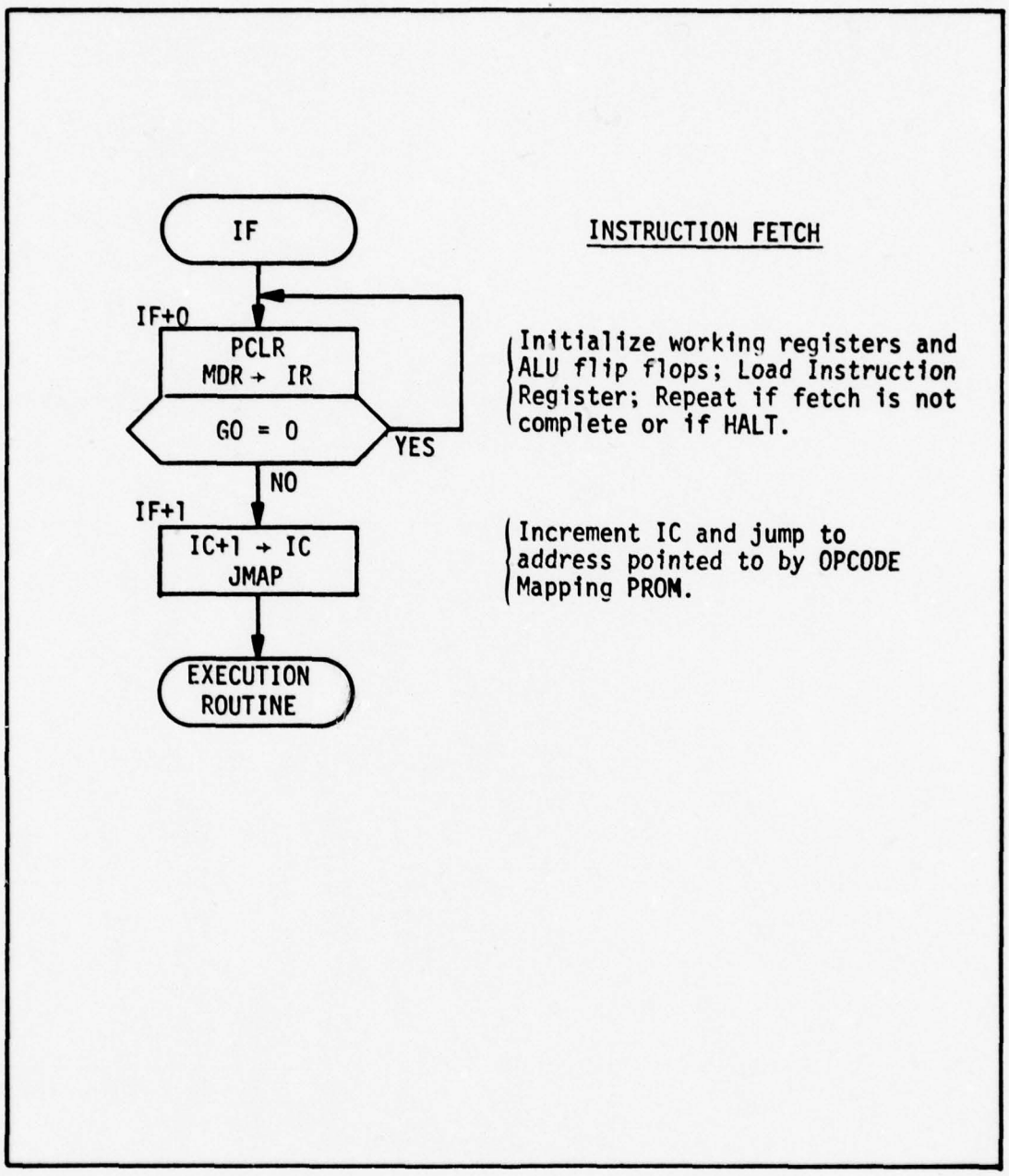


Fig. 29. Instruction Fetch Microroutine

instruction. This "prefetch" requirement increases throughput by overlapping the instruction fetch cycle with execution of the previous instruction.

The Instruction Fetch routine clears the processor's working registers and controls flip flops to prepare for a new machine instruction cycle. It completes the instruction fetch cycle (previously initiated) and loads the instruction into the Instruction Register (IR). If the GO/HALT switch is in the "Halt" position the IF routine will wait until the switch is moved to the "GO" position. Then it increments the Instruction Counter (IC) and transfers control to the Execution Routine pointed to by the Opcode Mapping PROM.

Execution. The Opcode Mapping PROM selects a unique Execution routine for each operation code. This routine interprets the machine instruction's address mode, fetches any required operands, and then executes the intended operation. Before the Execution routine terminates, it initiates a memory read to prefetch the next machine instruction and test for interrupt requests. If a valid interrupt request is pending, microprogram control is transferred to the Interrupt Handling microroutine. If a valid interrupt is not present, microprogram control returns to the Instruction Fetch Routine to begin a new instruction cycle.

An example of an Execution routine which shows the sequence of operations performed during the Double Precision Integer Add is presented on Figure 30. This instruction has three address modes which are identified by a different operation code. The microprogram subroutines which are used to decode these addressing modes will be described later.

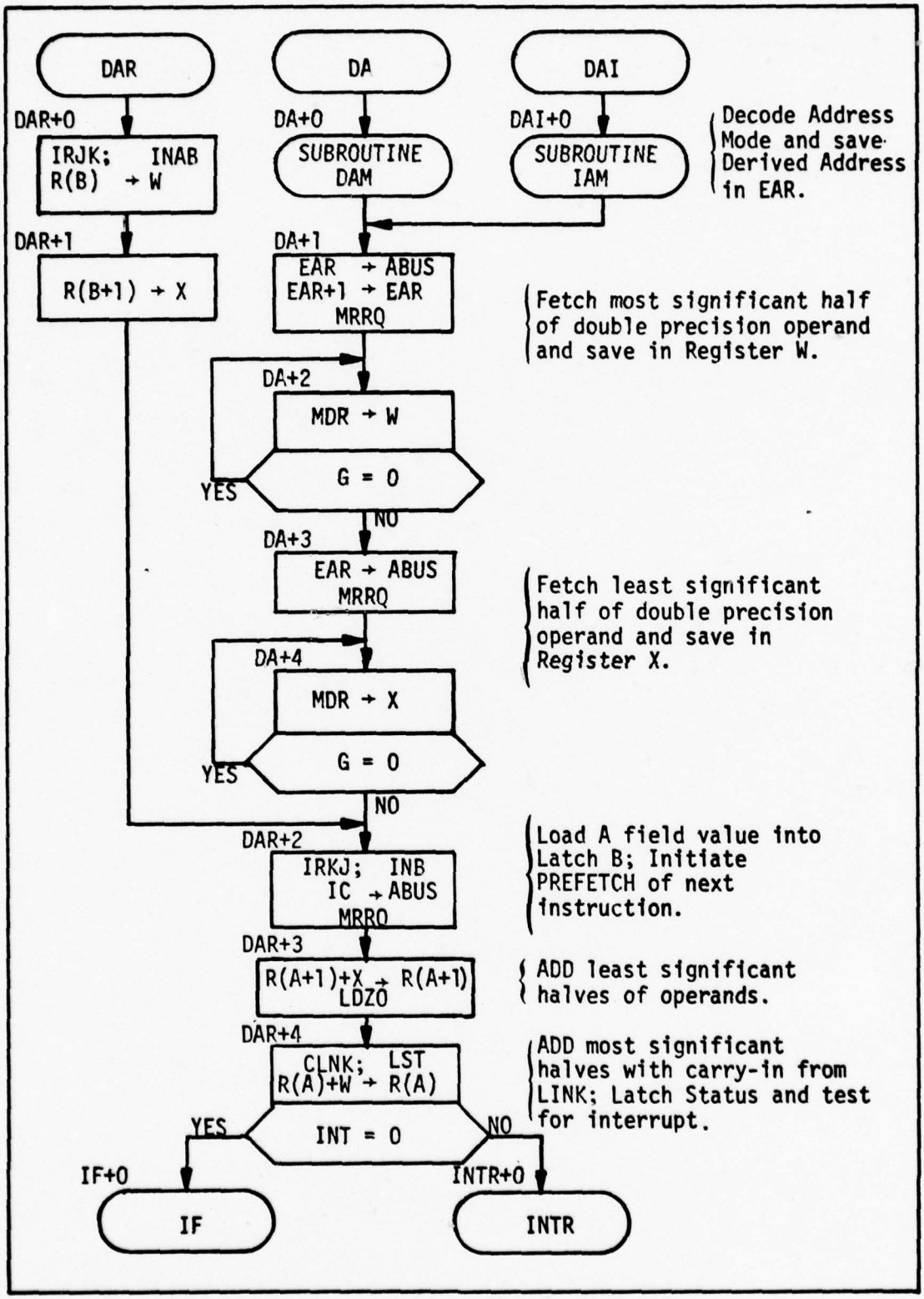


Fig. 30. Execution Microroutine Example
Double Precision Integer ADD

Interrupt Handling. The Interrupt Handling (INTR) microroutine is shown on Figure 31. This routine begins with a test of the EX flip flop. If that flip flop is set, the microroutine assumes that the previous instruction was executed from the derived address of an Execute (EX) instruction, and that a jump was not performed. In that case, the Interrupt Handling routine will ignore the instruction which was prefetched, load the IC with the address of the next sequential instruction following the EX instruction, and clear the EX flip flop. If the EX flip flop is not set, the Interrupt routine saves the old processor state and fetches the new processor state as described in Chapter II. Before the INTR routine transfers microprogram control to the Instruction Fetch routine, it initiates a memory read to prefetch the next machine instruction.

Address Mode Decoding

The address mode of each machine instruction is decoded by the instruction's Execution routine.

The task of decoding addressing modes is common to all execution routines. To simplify this task, a number of microprogram subroutines have been defined. These subroutines are shown on Figure 32.

Subroutines DAM, IAM, and ICR calculate the Derived Address for address modes D/DX, I/IX, and ICR (respectively). This address is saved in the EAR register. Subroutines IMAM, ISP and ISN calculate the Immediate operand for address modes IM/IMS, ISP, and ISN (respectively). This operand is saved in Register W.

The R and B modes are decoded without subroutines. The process of decoding the R address mode is too short to justify a special

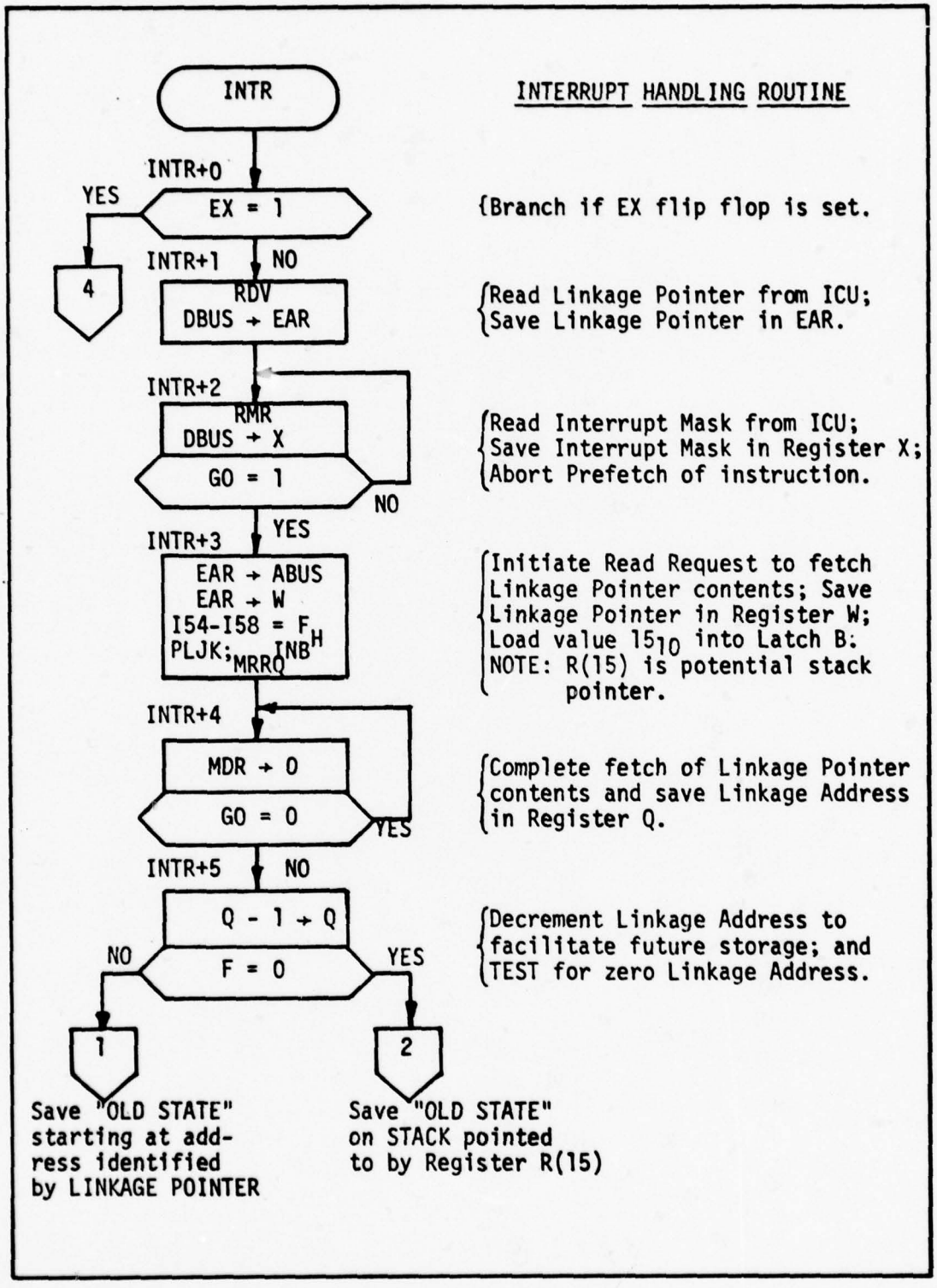


Fig. 31. Interrupt Handling Microroutine

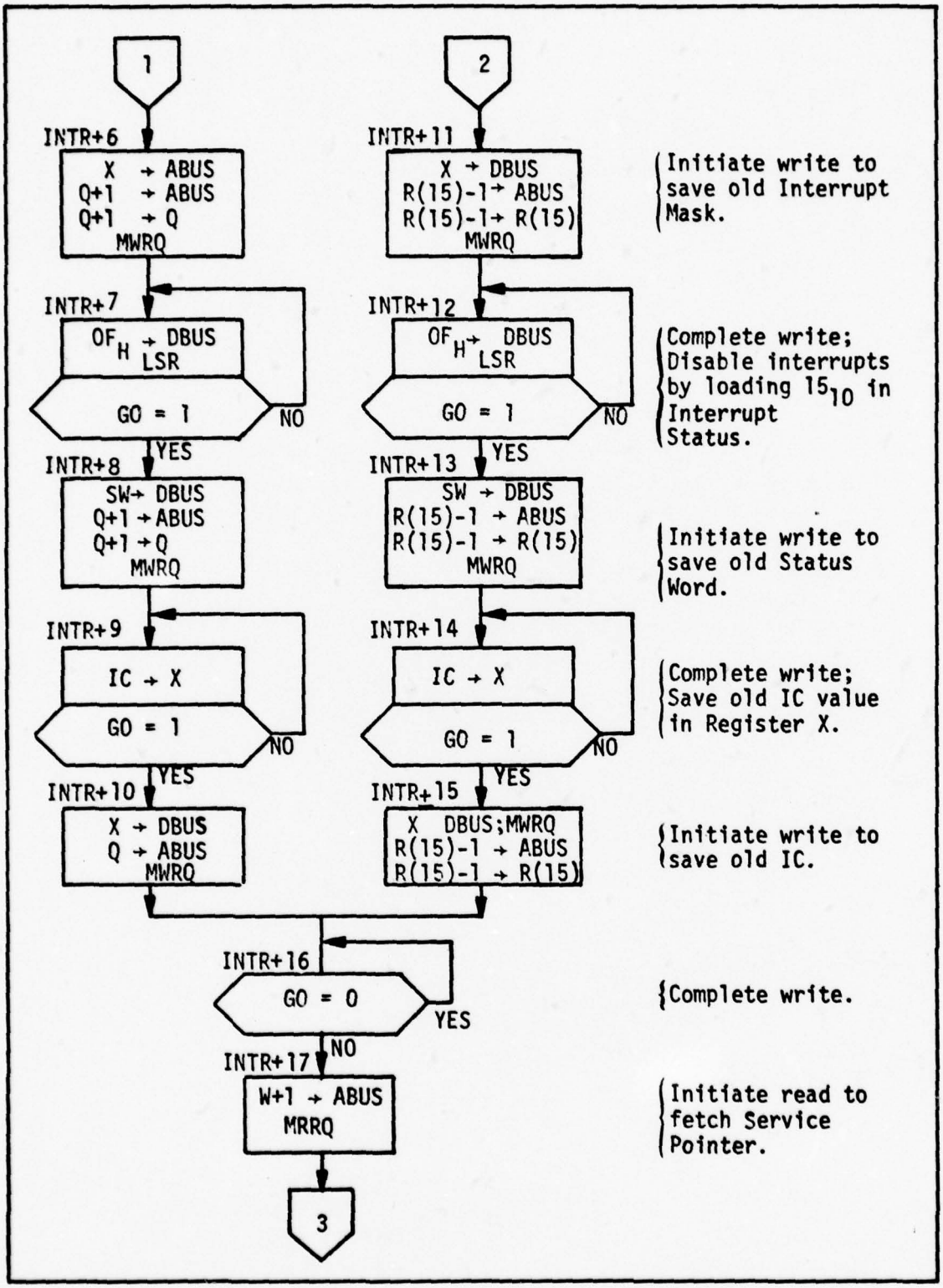


Fig. 31. Interrupt Handling Microroutine (continued)

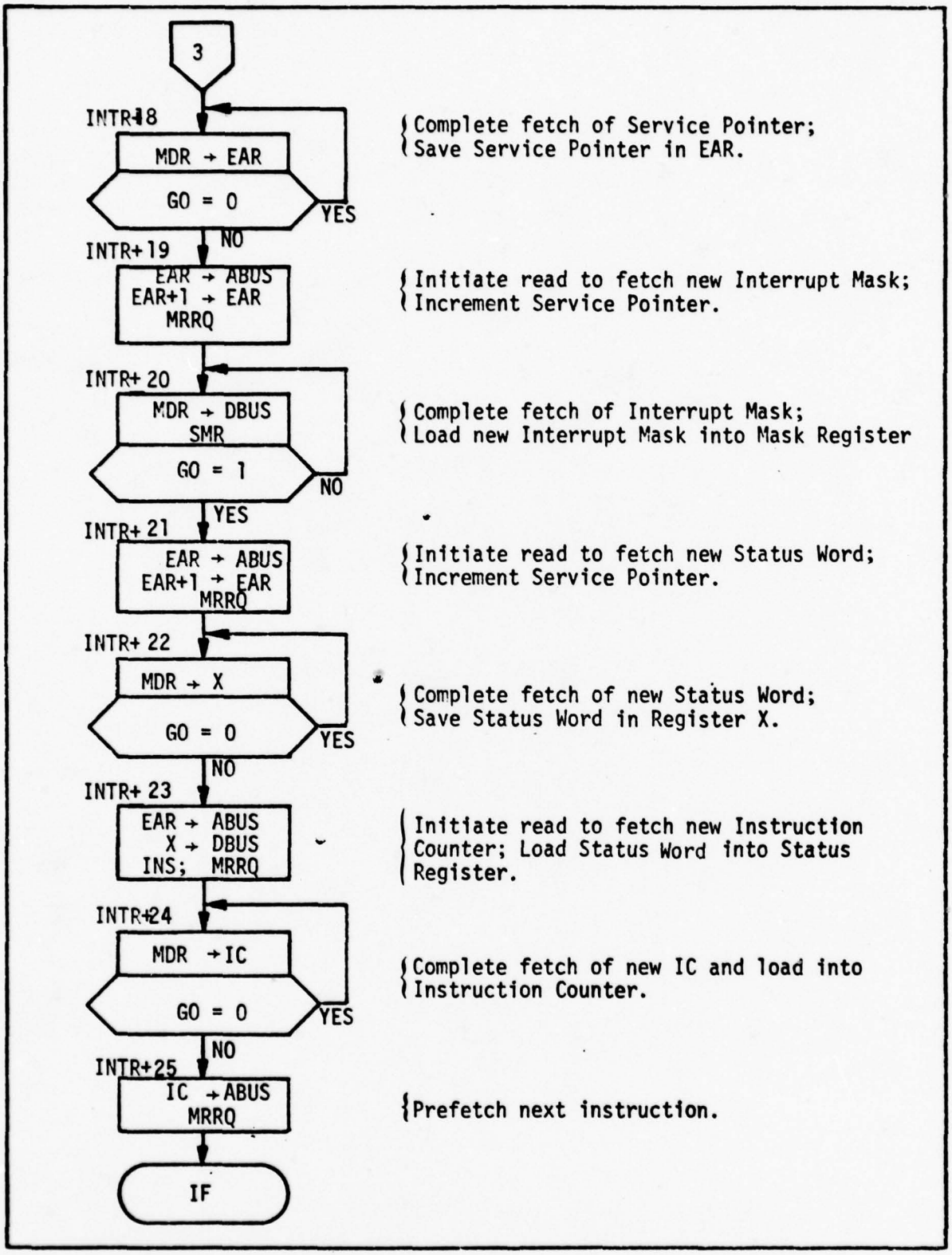


Fig. 31. Interrupt Handling Microroutine (continued)

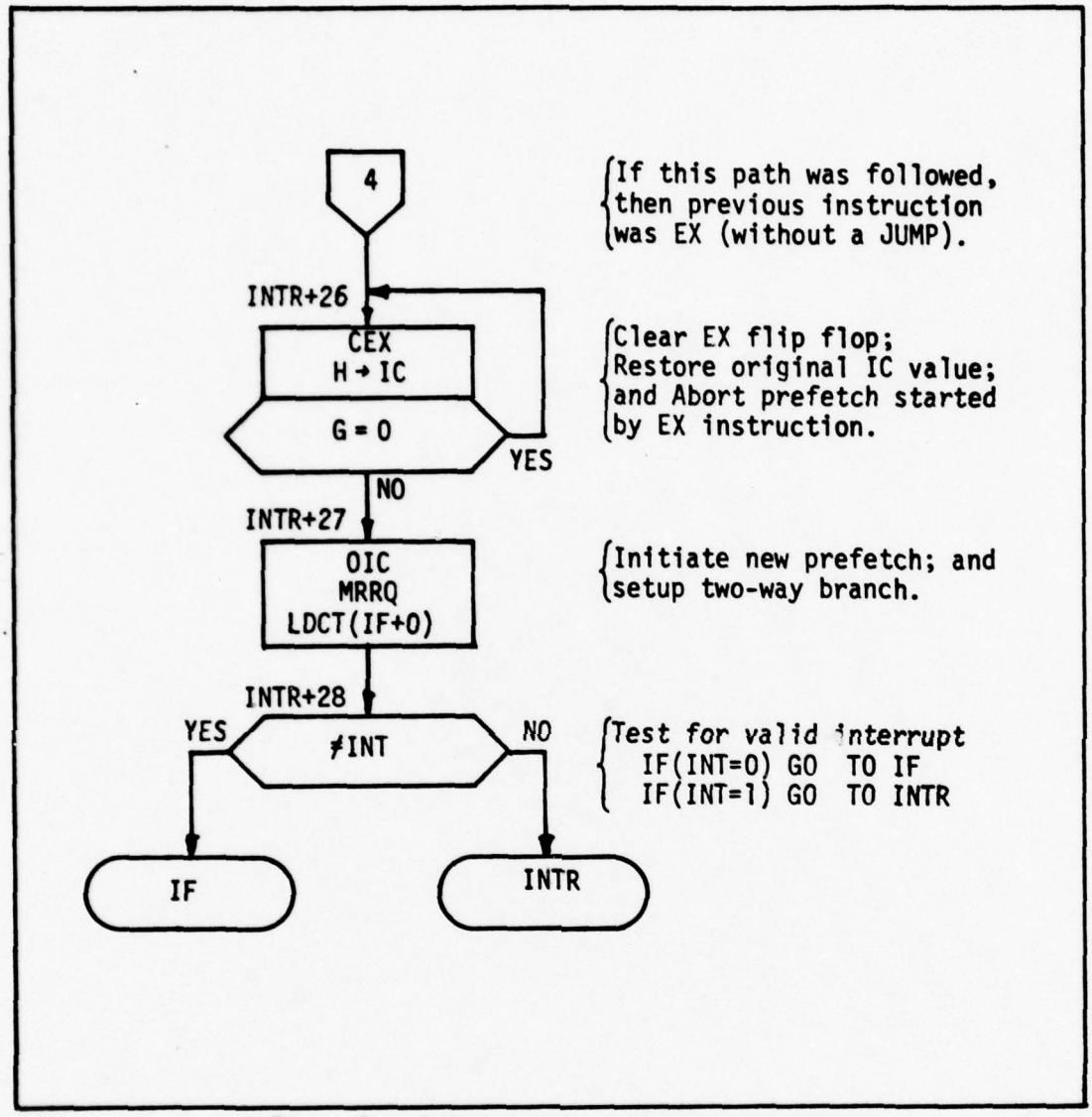


Fig. 31. Interrupt Handling Microroutine (continued)

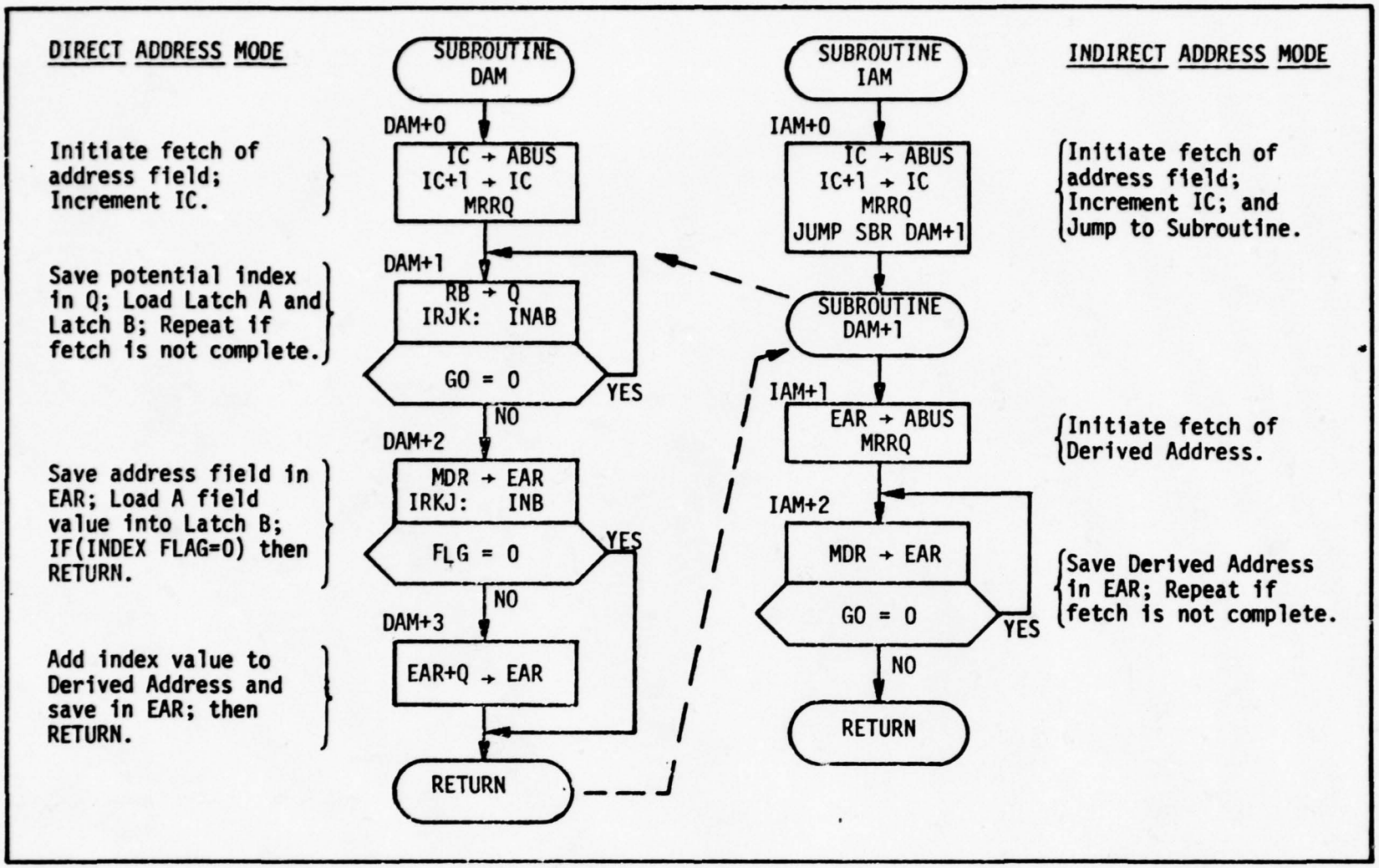


Fig. 32. Address Mode Subroutines

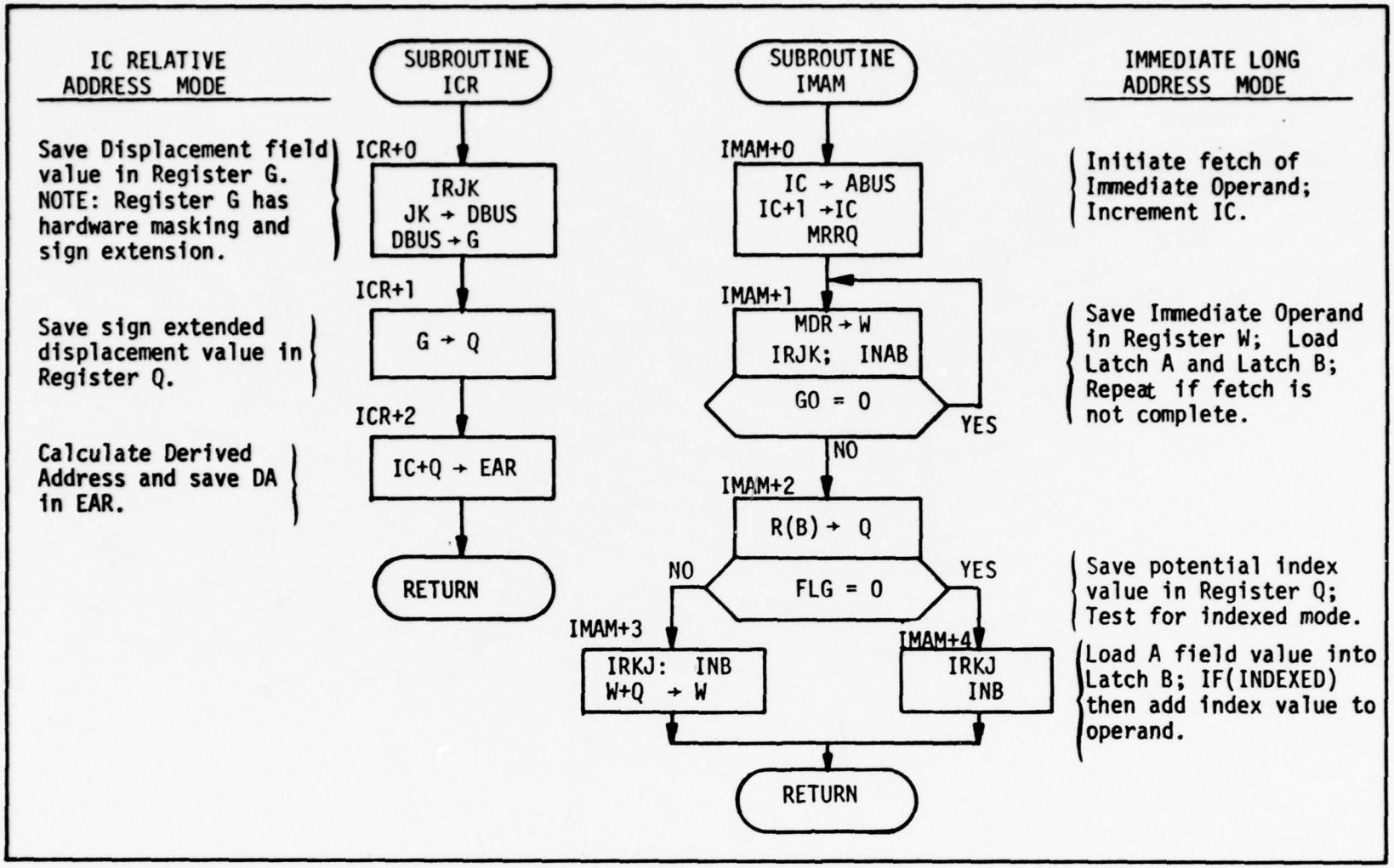


Fig. 32. Address Mode Subroutines (continued)

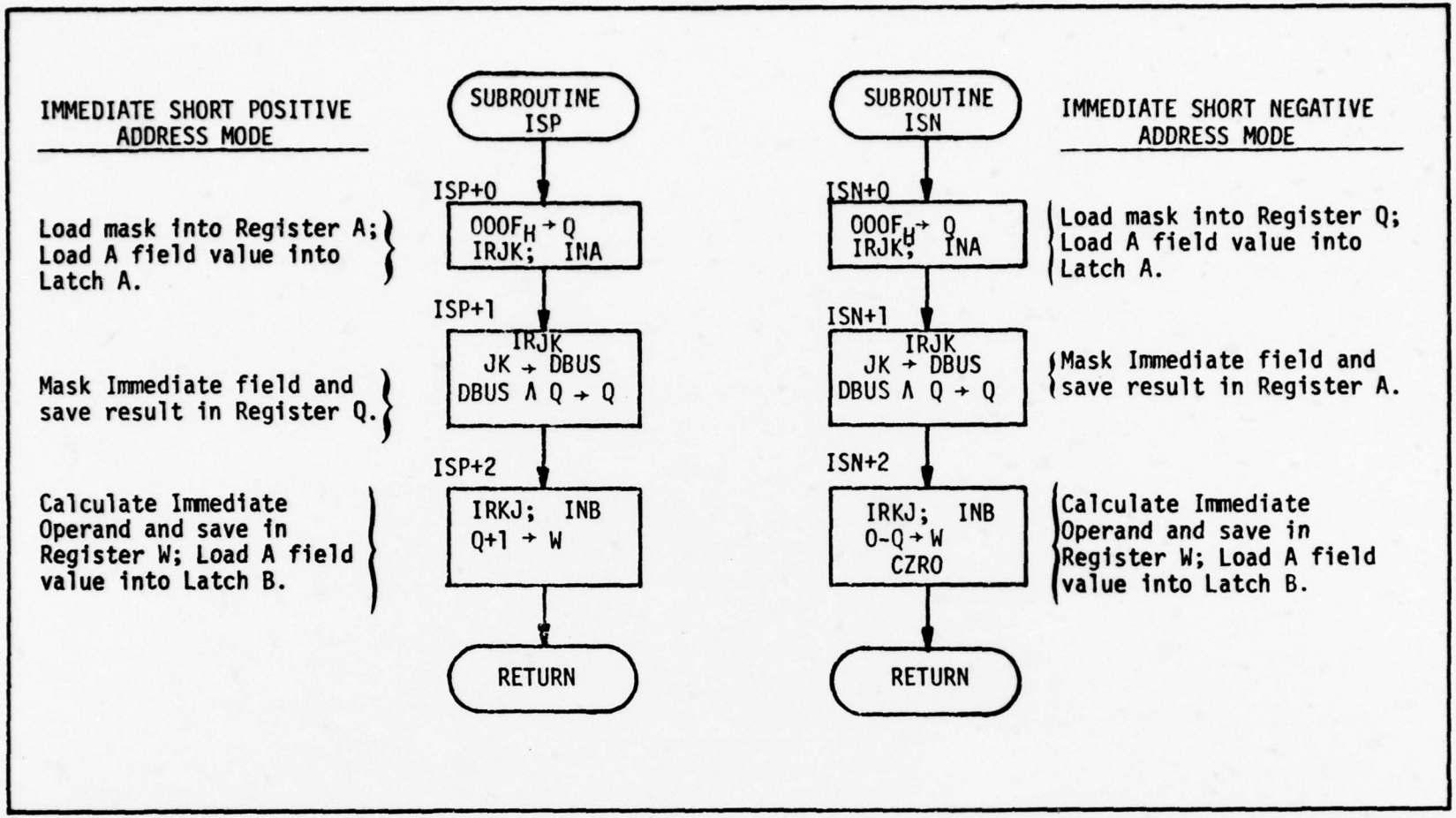


Fig. 32. Address Mode Subroutines (continued)

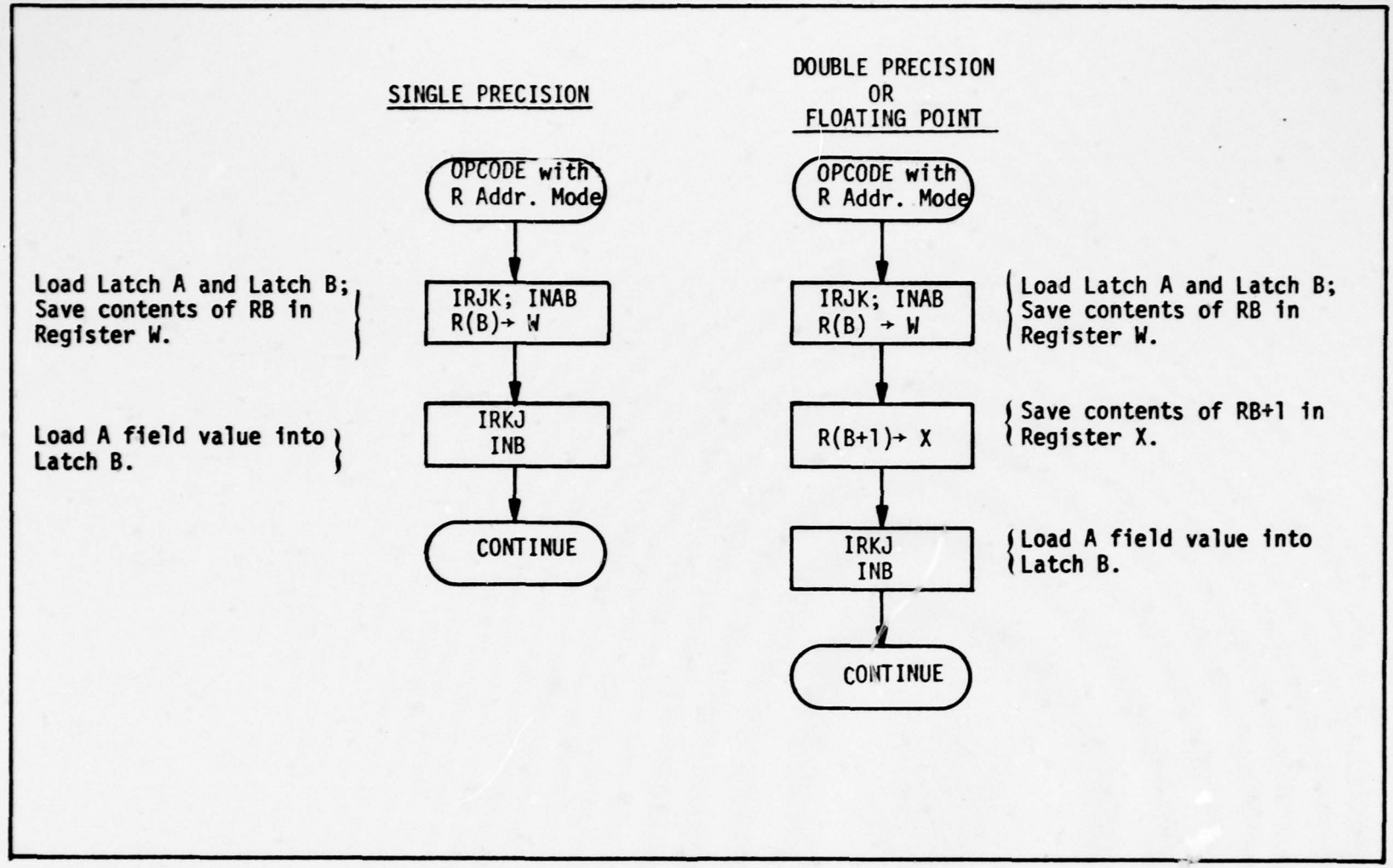


Fig. 33. Typical Decoding of R Address Mode

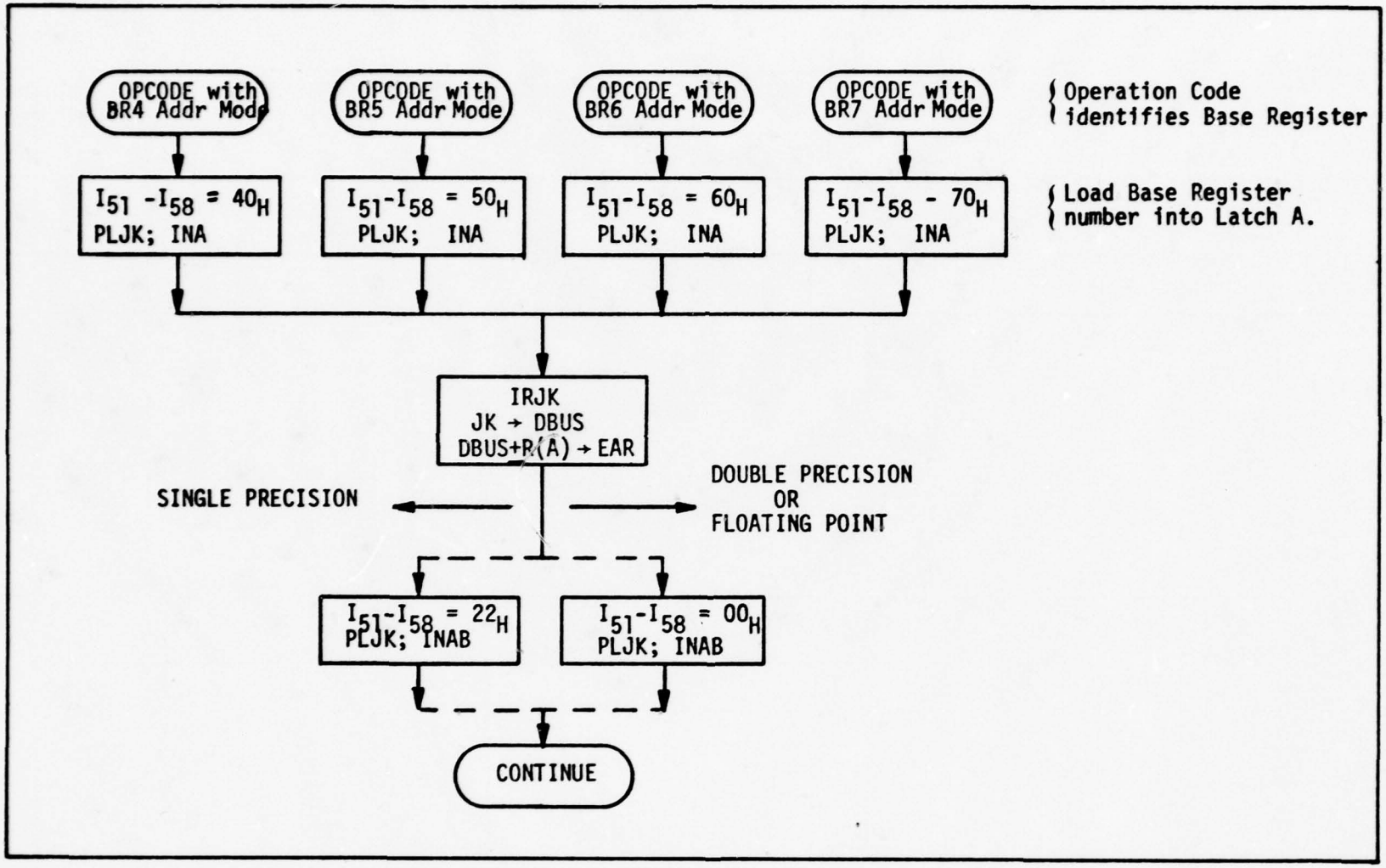


Fig. 34. Typical Decoding of B Address Mode

subroutine. A typical example of how the R-address mode is decoded is shown on Figure 33. The B address mode has too many variations to allow efficient decoding by a single subroutine. Each Base Relative instruction has four operation codes and each of these operation codes imply a different base register. In addition, single precision Base Relative instructions use Register R2 as the implied accumulator while double precision and floating point Base Relative instructions use Registers R0 and R1 as the implied accumulator. Figure 34 shows the typical method of decoding the B Address Mode.

Summary

This chapter introduced the processor microprogramming by describing the microprogram's control flow and principal microroutines. This chapter also described how instruction address modes are decoded. The efficiency with which this microprogram emulates the target instruction set will be discussed in the next chapter.

IX. Evaluation

This chapter discusses the operational speed of the processor and provides an estimate of the number of integrated circuit devices which are required to implement the design. The evaluations presented in this chapter will be used as the basis for the conclusions and recommendations in Chapter X.

Processor Speed

Instruction Execution Times. The execution times for each instruction in the software-compatible instruction set are given in Table VIII. These execution times were calculated with the assumptions that the processor's microinstruction cycle will be 250 nonoseconds (ns) and the access time for main memory will be less than 500ns (including I-BUS delay). The assumed 250ns microinstruction cycle is a conservative estimate which was selected to provide adequate timing margins for component tolerances when operating over the military temperature range of -55°C to 125°C. Likewise, the assumed 500ns main memory access time is also conservative when considering the availability of 250ns Random Access Memory.

Operational Speed. The processor's operational speed is 172 KOPS. This speed was determined by applying the instruction execution times, described above, to the Baseline Avionic Instruction mix as shown on Table IX. The Baseline Avionic Instruction Mix does not specify the address mode of some instructions. In those cases, the execution time listed on Table IX was obtained by averaging the execution times for all applicable address modes.

Table VIII

Instruction Execution Times

INSTRUCTION NAME	MNEMONIC	ADDRESS MODE	EXECUTION TIME	
			M CYCLES	u SECONDS
Single Precision Integer ADD	A	D,DX	14	3.5
	AR	R	5	1.25
	AI	I,IM	19	4.75
	AIM	IM	11	2.75
	AISP	ISP	9	2.25
	AISN	ISN	9	2.25
	AB	B	10	2.5
Double Precision Integer ADD	DA	D,DX	18	4.5
	DAR	R	7	1.75
	DAI	I,IX	23	5.75
Single Precision Integer SUBTRACT	S	D,DX	14	3.5
	SR	R	5	1.25
	SI	I,IX	19	4.75
	SIM	IM	11	2.75
	SBB	B	10	2.5
Double Precision Interger SUBTRACT	DS	D,DX	18	4.5
	DSR	R	7	1.75
	DSI	I,IX	23	5.75
Single Precision Integer MULTIPLY W 16-Bit Product	MS	D,DX	68	17.0
	MSR	R	60	15.0
	MSI	I,IX	73	18.25
	MSIM	IM	65	16.25
	MSIP	ISP	63	15.75
	MSIN	ISN	63	15.75
	MB	B	64	16.0
Single Precision Integer MULTIPLY W 32-Bit Product	M	D,DX	60	15.0
	MR	R	52	13.0
	MI	I,IX	65	16.25
	MIM	IM	57	14.25
Double Precision Integer MULTIPLY	DM	D,DX	151	37.75
	DMR	R	140	35.0
	DMI	I,IX	156	39.0
Single Precision Integer DIVIDE W 16-Bit Product	DV	D,DX	75	18.75
	DVR	R	67	16.75
	DVI	I,IX	80	20.0
	DVIM	IM	72	18.0
	DISP	ISP	70	17.5
	DISN	ISN	70	17.5
	DB	B	71	17.75

Table VIII

Instruction Execution Times

(continued)

INSTRUCTION NAME	MNEMONIC	ADDRESS MODE	EXECUTION TIME	
			M CYCLES	u SECONDS
Single Precision Integer DIVIDE 32-bit Dividend	D	D,DX	127	31.75
	DR	R	117	29.25
	DI	I,IX	132	33.0
	DIM	IM	125	31.25
Double Precision Integer DIVIDE	DD	D,DX	130	32.5
	DDR	R	119	29.75
	DDI	I,IX	135	33.75
INCREMENT Memory	IM	D,DX	19	4.75
DECREMENT Memory	DMM	D,DX	19	4.75
Single Precision ABSOLUTE Value	ABS	R	5	1.25
Double Precision ABSOLUTE Value	DABS	R	6	1.5
Single Precision NEGATE	NEG	R	5	1.25
Double Precision NEGATE	DNEG	R	6	1.5
Single Precision COMPARE	C	D,DX	15	3.75
	CR	R	7	1.75
	CI	I,IX	20	5.0
	CIM	IM	13	3.25
	CISP	ISP	11	2.75
	CISN	ISN	11	2.75
Double Precision COMPARE	DC	D,DX	17	4.25
	DCR	R	6	1.5
	DCI	I,IX	22	5.5
Floating Point COMPARE	FC	D,DX	23	5.75
	FCR	R	12	3.0
	FCI	I,IX	28	7.0
Floating Point ADD	FA	D,DX	52	13.0
	FAR	R	42	10.5
	FAI	I,IX	57	14.25
	FAB	B	49	12.25

Table VIII

Instruction Execution Times

(continued)

INSTRUCTION NAME	MNEMONIC	ADDRESS MODE	EXECUTION TIME	
			M CYCLES	u SECONDS
Floating Point SUBTRACT	FS	D,DX	52	13.0
	FSR	R	42	10.5
	FSI	I,IX	57	14.25
	FSB	B	49	12.25
Floating Point MULTIPLY	FM	D,DX	101	25.25
	FMR	R	91	22.75
	FMI	I,IX	106	26.5
	FMB	B	98	24.5
Floating Point ABSOLUTE Value	FABS	R	18	4.5
Floating Point NEGATE	FNEG	R	18	4.5
CONVERT Integer to Floating Point	FLT	R	23	5.75
CONVERT Floating Point to Integer	FIX	R	24	6.0
Logical AND	AND	D,DX	13	3.25
	ANDR	R	4	1.0
	ANDI	I,IX	18	4.5
	ANDM	IM	11	2.75
Inclusive Logical OR	OR	D,DX	13	3.25
	ORR	R	4	1.0
	ORI	I,IX	18	4.5
	ORIM	IM	11	2.75
Exclusive Logical OR	XOR	D,DX	13	3.25
	XORR	R	4	1.0
	XORI	I,IX	18	4.5
	XORM	IM	11	2.75
Logical NAND	N	D,DX	13	3.25
	NR	R	4	1.0
	NI	I,IX	18	4.5
	NIM	IM	11	2.75

Table VIII

Instruction Execution Times

(continued)

INSTRUCTION NAME	MNEMONIC	ADDRESS MODE	EXECUTION TIME	
			M CYCLES	u SECONDS
SHIFT Left	SLL SLC	R	6+1/shift	1.5+.25/shift
SHIFT Right	SRL SRA SRC	R	6+1/shift	1.5+.25/shift
Double SHIFT Left	DSLL DSLC	R	7+1/shift	1.75+.25/shift
Double SHIFT Right	DSRL DSRC DSRA	R	7+1/shift	1.75+.25/shift
SHIFT Count in Register	SRL SAR SCR	R	10+1/shift	2.5+.25/shift
Double SHIFT Count in Register	DSLR DSAR DSCR	R	11+1/shift	2.75+.25/shift
SET BIT	SB SBR SBI	D,DX R I,IX	15 4 20	3.75 1.0 5.0
RESET BIT	RB RBR RBI	D,DX R I,IX	15 4 20	3.75 1.0 5.0
SET Variable Bit	SVBR	R	5	1.25
RESET Variable Bit	RVBR	R	6	1.50
TEST Variable Bit	TVBR	R	5	1.25

Table VIII

Instruction Execution Times

(continued)

INSTRUCTION NAME	MNEMONIC	ADDRESS MODE	EXECUTION TIME	
			M CYCLES	u SECONDS
Single Precision LOAD	L	D,DX	14	3.5
	LR	R	4	1.0
	LI	I,IX	19	4.75
	LIM	IM,IMX	11	2.75
	LB	B	8	2.0
	LISP	ISP	9	2.25
	LISN	ISN	9	2.25
Double Precision LOAD	DL	D,DX	18	4.5
	DLR	R	5	1.25
	DLI	I,IX	23	5.75
	DLB	B	14	3.5
LOAD From Upper Byte	LUB	D,DX	20	5.0
	LUBI	I,IX	25	6.25
LOAD from Lower Byte	LLB	D,DX	20	5.0
	LLBI	I,IX	25	6.25
LOAD STATUS	LDST	D,DX	22	5.5
RETURN from Interrupt	RFI	R	18	4.5
LOAD Multiple Registers	LM	D,DX	11+4/load	2.75+1/load
Single Precision STORE	ST	D,DX	14	3.5
	STI	I,IX	19	4.75
	STB	B	11	2.75
Double Precision STORE	DST	D,DX	18	4.5
	DSTI	I,IX	23	5.75
	DSTB	B	15	3.75
Store Constant	STC	D,DX	16	4.0
	STCI	I,IX	21	5.25

Table VIII

Instruction Execution Times

(continued)

INSTRUCTION NAME	MNEMONIC	ADDRESS MODE	EXECUTION TIME	
			M CYCLES	u SECONDS
STORE Register Through Mask	SRM	D,DX	15	3.75
STORE into Upper Byte	STUB SUBI	D,DX I,IX	21 26	5.25 6.5
STORE into Lower Byte	STLB SLBI	D,DX I,IX	19 24	4.75 6.0
Move Multiple Words	MOV MOVI	D,DX I,IX	6+8/word 11+8/word	1.5+2/word 2.75+2/word
STORE Multiple Registers	STM	D,DX	13+4/store	3.25+1/word
PUSH Multiple Registers	PSHM	R	7+4/store	1.75+1/word
POP Multiple Registers	POPM	R	7+4/load	1.75+1/word
Stack IC and JUMP to Subroutine	SJS	D,DX	14	3.5
Unstack IC and RETURN from Subroutine	URTS	R	9	2.25
NO OPERATION	NOP	R	5	1.25
Break Point	BPT	R	Variable	Variable
EXECUTE	EX	D,DX	Variable	Variable
Jump Back and Count	JBC	D,DX	13	3.25

Table VIII

Instruction Execution Times

(continued)

INSTRUCTION NAME	MNEMONIC	ADDRESS MODE	EXECUTION TIME	
			M CYCLES	u SECONDS
BRANCH conditionally	BEZ	ICR	9	2.25
	BLT	ICR		
	BLE	ICR		
	BGT	ICR		
	BNZ	ICR		
	BGE	ICR		
BRANCH Always	BR	ICR	6	1.5
JUMP to Subroutine	JS	D,DX	11	2.75
Jump on Condition	JC	D,DX	9	2.25
	JCI	I,IX	11	2.75
Console Input/Output	CIO	IM	5	1.25
INPUT	ITA	IM,IMX	16	4.0
	ITB		16	4.0
	RSW		16	4.0
	ID		16	4.0
	RIC		16	4.0
	PI		18	4.5
OUTPUT	GO	IM,IMX	16	4.0
	OTA		16	4.0
	OTB		16	4.0
	DMAE		16	4.0
	DMAD		16	4.0
	DSBL		16	4.0
	OD		16	4.0
	CLIR		16	4.0
	SIC		16	4.0
	ENBL		16	4.0
	PO		18	4.5

Table IX

Baseline Avionic Instruction Mix

INSTRUCTION TYPE	PERCENT		EXECUTION TIME		
Floating Point:					
Add/Subtract	14	x	12.5 usec	=	1.75
Multiply	5	x	24.75	=	1.24
Divide	1	x	37.0	=	.37
Load/Store	10	x	3.75	=	.38
Fixed Point:					
Add/Subtract:					
Memory Direct	1	x	3.5	=	.04
Immediate	2	x	2.5	=	.05
Memory Direct Indexed	1	x	3.75	=	.04
Load:					
Memory Direct	22	x	3.5	=	.77
Memory Direct Indexed	3	x	3.75	=	.11
Register Direct	4	x	1.0	=	.04
Store	9	x	3.5	=	.32
Add Register Direct	1	x	1.25	=	.01
Decrement and Branch	9	x	3.25	=	.29
Conditional Branch	18	x	2.25	=	.41
			Weighted Average=		5.82usec

$$\text{OPERATIONAL SPEED} = \frac{1}{5.82\text{usec}} = 172 \text{ KOPS}$$

Discussion. Inspection of Table IX indicates that the processor's operational speed is severely restricted by its slow execution of floating point instructions. Considering the floating point instructions and fixed point instructions separately, the processors operational speed is 338 KOPS. This observation suggests that the processor's operational speed could be significantly increased by adding hardware to facilitiate floating point operations.

A second factor which significantly reduces the processor's operational speed is the overhead required to synchronize I/O transfers over the I-BUS. The processor requires 4 microinstruction cycles (1 micro second) to accomplish each I-BUS transfer. One microcycle is required to initiate the transfer, at least two microcycles (500ns) are required to allow for memory access time, and a fourth microcycle is required to verify completion of the transfer. The adverse effect of this I/O overhead is reduced, in some cases, by the autonomous nature of the IOIU which permits the ALU to perform useful calculations while an I/O transfer is in progress. However, during I/O intensive operations such as address mode decoding or interrupt servicing there is little opportunity to exploit this potential parallelism. The Interrupt Handling Routine, shown on Fig. 31, illustrates the adverse effects of the I/O overhead. It requires 33 microinstruction cycles (9.5 micro seconds) to accomplish 9 Input/Output transfers.

Parts Count

Table X shows that approximately 170 integrated circuit devices are required to implement the processor's design. This estimate was obtained by dividing the design into subunits and estimating the number

Table X

Estimated Parts Count

Processor Component	Number of IC's
1. Arithmetic and Logic Unit	
Computation Unit	7
Shift Multiplexers	2
External Registers	23
Flip Flops	6
Misc.	6
2. Computer Control Unit	
Microsequencer (AM 2910)	1
Control Memory (INTEL 3604A-2)	24
Pipeline Register	8
OP Code Mapping Prom (INTEL 3621-1)	3
I/O Mapping Prom (AM 29751)	2
Decoders	4
Registers/Counters	12
Multiplexers	4
Buffers	6
Misc.	6
3. Interrupt Control Unit	
Interrupt Encoders (AM 2914)	2
Buffers	3
Misc.	2
4. I/O Interface Unit	
I-BUS Interface:	
Flip Flops	2
Latches/Registers	6
Sequence Controller	7
DMA Controller	5
Misc.	12
Discrete Data Interface	4
Interval Timers	7
Control Panel	6
TOTAL	170

of devices required for each subunit. The processor's part count (Table X) is discussed in following paragraphs to identify factors in the processor's specifications which effect the parts count.

The estimated device count for the Arithmetic and Logic Unit is 44 integrated circuits. It is significant to note that over half of these devices are associated with the ALU's external registers. The requirement for 16 general purpose registers committed the entire 16 word general register file of the ALU to the user. With the general register file committed, it was necessary to add external registers to provide temporary operand storage and to accommodate program control functions.

The large size of the microprogram required to emulate the software-compatible instruction influences the device count of the Computer Control Unit (CCU). The estimated parts count for the CCU is 70 integrated circuit devices. Of these 70 devices, approximately 24 will be used in the control memory to store the emulation microprogram (1233 microinstruction words).

Summary

This chapter described the execution times for each instruction in the target instruction set and discussed the processor's operational speed. It also presented an estimate of the number of integrated circuit devices which are required to realize the processor's design. The information presented in this chapter will be used as the basis for conclusions and recommendations presented in the next chapter.

X. Conclusions and Recommendations

This chapter presents conclusions and gives recommendations for improving the processor and for refining the specifications of the software-compatible avionic computer family.

Conclusions

The processor designed in this report emulates all instructions of the baseline software-compatible instruction set. It fulfills the basic architectural requirements; but, it does not satisfy the operational speed requirement. The estimated speed of the processor (172 KOPS) falls below the required operational speed range of 200 to 500 KOPS.

The processor's slow operational speed is not directly related to any inherent limitations in the AM 2900 device family or any deficiencies in the specifications of the software-compatible avionic computer family. Evaluation of the design identified two factors which restricted the processor's speed. First, the decision to implement floating point arithmetic using microprogram software (rather than hardware) resulted in comparatively slow execution speeds for floating point operations. Second, the design of the processor Input/Output Interface Unit relies upon the microprogram software to synchronize Input/Output operations over the processor's I-BUS. This synchronization overhead reduces the processor's speed in fetching instructions, operands, and data.

The addressing modes and instruction formats used in the software-compatible instruction set were easily interpreted by the processor's hardware, and emulation of these instructions presented no unusual problems. However, evaluation of the processor's design showed that the

overall size and complexity of the instruction set influenced the parts count indirectly by requiring large control memory (1233 microinstruction words).

Recommendations

Design Improvements. The processor's operational speed can be increased by adding floating point hardware to the processor's Arithmetic and Logic Unit. This hardware could be directed at reducing the processor's dependence on software for exponent comparisons, testing for exponent overflow/underflow and additions, and normalizing floating point operands.

The processor's speed is also restricted by the overhead required to synchronize I-BUS transfers. This overhead can be reduced by redesigning the processor's Input/Output Interface Unit to allow pipelining of I-BUS transfers. This would require additional buffers to the IOIU so that one transfer could be "set up" while the previous transfer was still in progress.

Interrupt Changes. The interrupt system defined for the software-compatible avionic computer family requires nine I-BUS transfers to service an interrupt request. It is recommended that the AFAL investigate the processor's interrupt environment to determine if potential application can tolerate this overhead. One suggestion for increasing interrupt speed is to reduce the number of I-BUS transfers by allowing optional storage and retrieval of the Interrupt Mask. This option could be flagged by a bit in the processor's status word. A second suggestion is to eliminate the requirement for fetching the Linkage Pointer contents

by always storing the old computer state on a stack which is pointed to by an implied stack pointer (R15).

Comments

Advanced Micro Device has announced the development of the AM 2903 bipolar microprocessor slice which is expected to be commercially available by the middle of 1978. This new device performs all the functions of the AM 2901A and provides a number of enhancements which could significantly improve the processor's performance and reduce the processor's parts count. (Ref 6:1).

Bibliography

1. Air Force Systems Command. Required Features of the Microcomputer. Solicitation No. F33615-77-R-1176. Wright-Patterson Air Force Base, Ohio: Air Force Avionics Laboratory, April 1977.

2. Air Force Systems Command. Distributed Processor/Memory Architecture Design Program. AFAL-TR-75-80. Wright-Patterson Air Force Base, Ohio: Air Force Avionics Laboratory, February 1975.

3. Advanced Micro Devices Incorporated. AM2900 Bipolar Microprocessor Family Data Book. Sunnyvale, California: 1976.

4. Advanced Micro Devices Incorporated. AM2910 Microprogram Controller Preliminary Data. Sunnyvale, California: 1977.

5. Advanced Micro Devices Incorporated. AM2914 Vectored Priority Interrupt Controller. Sunnyvale, California: 1976.

6. Coleman, Vernon, Michail W. Economido, and William J. Harmon, Jr. The Next Generation Four-Bit Bipolar Microprocessor Slice -- The AM 2903. Sunnyvale, California: Advanced Micro Devices, 1977.

Appendix A

Instruction Set Description

Table A1 is a matrix of the microcomputer instructions. Each instruction mnemonic and its corresponding operation code are specified by the table. The horizontal axis represents the more significant hexadecimal digit of the operation code and the vertical axis represents the less significant digit. Table A2 describes the instruction word format for each address mode. Table A3 describes the derived operand (DO) and derived address (DA) associated with each of the various addressing modes. Table A4 provides an abbreviated description of each microcomputer instruction.

The following abbreviations are used throughout this appendix to describe the instruction set.

Data Quantities:

MSH - - - - - - - - - - - Most Significant Half.

LSH - - - - - - - - - - - Least Significant Half.

MSB - - - - - - - - - - - Most Significant Bit

LSB - - - - - - - - - - - Least Significant Bit

s.p.- - - - - - - - - - - Single Precision

d.p.- - - - - - - - - - - Double Precision

f.p.- - - - - - - - - - - Floating Point

Addressing Modes (Refer Table A2):

R - - - - - - - - - - - - Register Direct

D,DX- - - - - - - - - - - Memory Direct, or Memory Direct Indexed.

I,IX- - - - - - - - - - - Memory Indirect, or Memory Indirect with Pre-Indexing

IM,IMX- - - - - - - - - - Immediate Long, or Immediate Long with Indexing

ISP - - - - - - - - - - - Immediate Short with Positive Operand.

ISN - - - - - - - - - - - Immediate Short with Negative Operand.

ICR - - - - - - - - - - - IC Relative.

B - - - - - - - - - - - - Base Relative.

CPU Registers:

R0, R1, ... , R15 - - - The 16 (16-bit) general purpose registers.

RA - - - - - - - - - - - General purpose register containing the required operand specified by the instruction.

(RA,RA+1) - - - - - - - - Concatenated pair of general purpose registers containing a 32-bit operand. Register RA, specified by the instruction, must be an even numbered register and contains the MSH of the operand.

RB - - - - - - - - - - - General purpose register containing the DO for R addressing mode instructions.

(RB,RB+1) - - - - - - - - Concatenated pair of general purpose registers containing a 32-bit operand for the R addressing mode. RB is an even numbered register and stores the MSH of the 32-bit operand.

RX - - - - - - - - - - - General purpose register used as an index register for DX and IX addressing modes.

BR - - - - - - - - - - - General purpose register used as the base register for B addressing mode.

Other:

OC - - - - - - - - - - - Operation Code.

OXC - - - - - - - - - - - Operation Code Extension.

DO - - - - - - - - - - - Derived Operand (Refer Table A3).

DA - - - - - - - - - - - Derived Address (Refer Table A3).

{X} - - - - - - - - - - - Contents of memory location x.

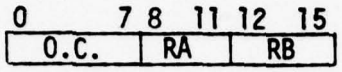
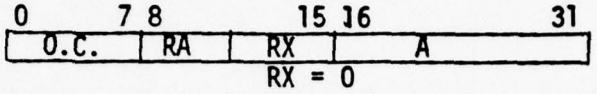
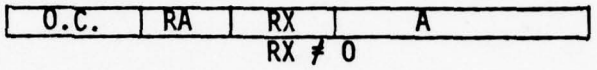
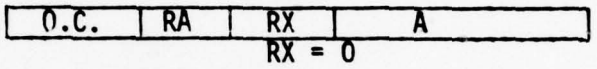
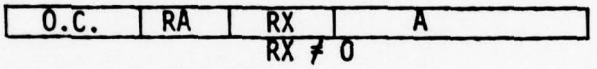
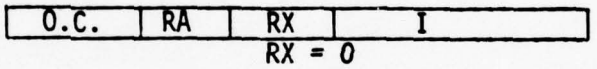
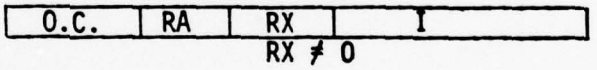
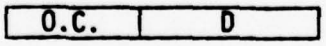
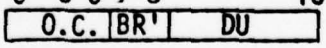
(X) - - - - - - - - - - - Contents of register X.

Table A1
Op Code Matrix

MOST SIGNIFICANT HEX DIGIT

LEAST SIGNIFICANT HEX DIGIT	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LB BR4	AB BR4	FAB BR4	BEZ		SB	SLL	JC	L	ST	A	S	M	D	OR	C
1	LB BR5	AB BR5	FAB BR5	BLT		SBR	SRL	JCI	LR	STC	AR	SR	MR	DR	ORR	CR
2	LB BR6	AB BR6	FAB BR6	BLE		SBI	SRA	JS	LI	STI	AI	SI	MI	DI	ORI	CI
3	LB BR7	AB BR7	FAB BR7	BGT		RB	SLC		LIM	STCI	AIM	SIM	MIM	DIM	ORIM	CIM
4	DLB BR4	SBB BR4	FSB BR4	BNZ		RBR	SRC	JBC	DL	DST	DA	DS	DM	DD	AND	DC
5	DLB BR5	SBB BR5	FSB BR5	BGE		RBI	DSLL	BR	DLR	SRM	DAR	DSR	DMR	DDR	ANDR	DCR
6	DLB BR6	SBB BR6	FSB BR6	CISP		TB	DSRL	EX	DLI	DSTI	DAI	DSI	DMI	DDI	ANDI	DCI
7	DLB BR7	SBB BR7	FSB BR7	CISN		TBR	DSRC	BPT	LUB	STUB	FA	FS	FM	FD	ANDM	FC
8	STB BR4	MB BR4	FMB BR4	LISP		TBI	DSLC		LLB	STLB	FAR	FSR	FMR	FDR	XOR	FCR
9	STB BR5	MB BR5	FMB BR5	LISN			DSRA	CIO	PSHM	POPM	FAI	FSI	FMI	FDI	XORR	FCI
A	STB BR6	MB BR6	FMB BR6	AISP		SVBR	SLR	URTS	IN	OUT	IM	DMM	MS	DV	XORI	
B	STB BR7	MB BR7	FMB BR7	AISN			SAR	SJS	RFI		ABS	NEG	MSR	DVR	XORM	
C	DSTB BR4	DB BR4	FDB BR4	MISP		RVBR	SCR	LUBI	FIX	MOV	DABS	DNEG	MSI	DVI	N	
D	DSTB BR5	DB BR5	FDB BR5	MISN			DSLR	LLBI	FLT	MOVI	FABS	FNEG	MSIM	DVIM	NR	
E	DSTB BR6	DB BR6	FDB BR6	DISP		TVBR	DSAR	SUBI	LDST						NI	
F	DSTB BR7	DB BR7	FDB BR7	DISN			DSCR	SLBI	LM	STM					NIM	NOP

Table A2
Instruction Word Format

	ADDRESSING MODE	SYMBOL	FORMAT
1.	Register Direct	R	
2.	Memory Direct Non-Indexed	D	
3.	Memory Direct Indexed	DX	
4.	Memory Indirect Non-Indexed	I	
5.	Memory Indirect Indexed	IX	
6.	Immediate Long Non-Indexed	IM	
7.	Immediate Long Indexed	IMX	
8.	Immediate Short Positive	ISP	
9.	Immediate Short Negative	ISN	
10.	IC-Relative	ICR	
11.	Base Relative	B	

BR = BR' + 4
Implied Accumulator:
F.P. = (R0, R1)
D.P. = (R0, R1)
S.P. = R2

Table A3
Address Mode Description
Derived Operand and Derived Address

ADDRESS MODE	SYMBOL	DERIVED OPERAND (DO)		DERIVED ADDRESS (DA)	
		S.P.	Flt.P. & D.P.	S.P.	Flt.P. & D.P.
Register Direct	R	(RB)	(RB,RB+1)	RB	RB,RB+1
Memory Direct:					
Non-Indexed	D	{A}	{A,A+1}	A	A,A+1
Indexed	DX	{A+(RX)}	{A+(RX),A+1+(RX)}	A+(RX)	A+(RX),A+(RX)+1
Memory Indirect:					
Non-Indexed	I	{{A}}	{(A),(A)+1}	{A}	{A},{A}+1
Indexed	IX	{{A+(RX)}}	{{A+(RX)},{A+(RX)+1}}	{A+(RX)}	{A+(RX)},{A+(RX)+1}
Immediate Long					
Non-Indexed	IM	I	-	-	-
Indexed	IMX	I+(RX)	-	-	-
Immediate Short					
Positive	ISP	+(I+1)	-	-	-
Negative	ISN	-(I+1)	-	-	-
IC-Relative	ICR	-	-	D+(IC)	-
Base Relative	B	{DU+(BR)}	{DU+(BR),DU+1+(BR)}	DU+(BR)	DU+(BR),DU+1+(BR)

Table A4
Instruction Set Description

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Single Precision Integer ADD	A0 A1 A2 A3 3A 3B 10-13	A AR AI AIM AISP AISN AB	D,DX R I,IX IM ISP ISN B	The DO is added to the contents of RA. The result, a 2's complement sum, is stored in RA. Status is set based on the result in RA and overflow.
Double Precision Integer ADD	A4 A5 A6	DA DAR DAI	D,DX R I,IX	The d.p. DO is added to the d.p. contents of (RA,RA+1). The result, a 2's complement 32-bit sum, is stored in (RA,RA+1). Status is set based on the result in (RA,RA+1) and overflow. NOTE: RA(RB) must be an even numbered register.
Single Precision Integer SUBTRACT	B0 B1 B2 B3 14 - 17	S SR SI SIM SBB	D,DX R I,IX IM B	The DO is subtracted from the contents of RA. The result a 2's complement difference, is stored in RA. Status is set based on the result in RA and overflow.
Double Precision Integer SUBTRACT	B4 B5 B6	DS DSR DSI	D,DX R I,IX	The d.p. DO is subtracted from the d.p. contents of (RA,RA+1). The result, a 2's complement 32-bit difference, is stored in (RA,RA+1). Status is set based on the result in (RA,RA+1) and overflow. NOTE: RA(RB) must be an even numbered register.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Single Precision Integer MULTIPLY with 16-bit Product	CA CB CC CD 3C 3D 18 - 1B	MS MSR MSI MSIM MISP MISN MB	D,DX R I,IX IM ISP ISN B	The DO is multiplied by the contents of RA. The LSH of the result, a 2's complement product, is stored in RA. Status is set based on the result in RA and overflow. Overflow occurs if the MSH of the product is not zero.
Single Precision Integer MULTIPLY with 32-bit Product	C0 C1 C2 C3	M MR MI MIM	D,DX R I,IX IM	The DO is multiplied by the contents of RA. The result, a 32-bit 2's complement product, is stored in (RA,RA+1) with the MSH of the product in RA. Status is set based on the result in (RA,RA+1). *NOTE: RA(RB) must be an even numbered register.*
Double Precision Integer MULTIPLY	C4 C5 C6	DM DMR DMI	D,DX R I,IX	The d.p. DO is multiplied by the d.p. contents of (RA,RA+1). The LSH of the result, a 32-bit 2's complement product, is stored in (RA,RA+1) with MSH in RA. Status is set based on the result in (RA,RA+1) and overflow. Overflow occurs if the MSH of the product is not zero. NOTE: RA(RB) must be an even numbered register.
Single Precision Integer DIVIDE with 16-bit Dividend	DA DB DC DD 3E 3F 1C - 1F	DV DVR DVI DVIM DISP DISN DB	D,DX R I,IX IM ISP ISN B	The contents of RA are divided by the DO. The result is stored in RA and RA+1 such that RA stores the s.p. quotient and RA+1 stores the remainder. Status is set based on the result in RA and overflow. Overflow occurs if the divisor is zero. NOTE: RA(RB) must be an even numbered register. : The sign of the remainder is the same as the sign of the dividend.

Table A 4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Single Precision Integer DIVIDE with 32-bit Dividend	D0 D1 D2 D3	D DR DI DIM	D,DX R I,IX IM	The d.p. contents of (RA,RA+1) is divided by the D0, a s.p. 2's complement number. The result is stored in (RA,RA+1) such that RA stores the s.p. integer quotient and RA+1 stores the remainder. Status is set based on the result in RA and overflow. Overflow occurs if the divisor equals zero or if the magnitude of the MSH of the dividend is greater than or equal to the magnitude of the divisor. NOTE: RA(RB) must be an even numbered register. : The sign of the remainder is the same as the sign of the dividend.
Double Precision Integer DIVIDE	D4 D5 D6	DD DDR DDI	D,DX R I,IX	The d.p. contents of (RA,RA+1) is divided by the d.p. D0. The quotient part of the integer result is stored in (RA, RA+1) and the remainder is lost. Status is set based on the result in (RA,RA+1) and overflow. Overflow occurs if the divisor is zero. NOTE: RA(RB) must be an even numbered register.
INCREMENT Memory by a Positive Integer	AA	IM	D,DX	This instruction adds a positive constant to a memory location. The contents of the memory location specified by the DA is incremented by N, where N is an integer ($1 \leq N \leq 16$) which is obtained by adding one to the "RA" field of the instruction. Status is set based on the result of the addition and overflow.
DECREMENT Memory by a Positive Integer	BA	DMM	D,DX	This instruction subtracts a positive constant from a memory location. The contents of the memory location specified by the DA is decremented by N, where N is an integer ($1 \leq N \leq 16$) which is obtained by adding one to the "RA" field of the instruction. Status is set based on the result of the subtraction and overflow.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Single Precision ABSOLUTE Value of Register	AB	ABS	R	If the sign bit of the D0 (RB) is one, the negative or 2's complement of the D0 is stored into RA. However, if the sign bit of the D0 is zero, the D0 is stored, unchanged, into RA. Status is set based on the result in RA. NOTE: RA may equal RB.
Double Precision ABSOLUTE Value of Register	AC	DABS	R	If the sign bit of the d.p. D0 (RB,RB+1) is one, the negative or 2's complement of the D) is stored into (RA,RA+1). However, if the sign bit of the D0 is zero, the D0 is stored, unchanged, into (RA,RA+1). Status is set based on the result in (RA,RA+1). NOTE: RA(RB) must be an even numbered register. : RA may equal RB.
Single Precision NEGATE Register	BB	NEG	R	The negative (2's complement) of the D0 (RB) is stored into RA. Status is set based on the result in RA. NOTE: The negative of zero is zero. : RA may equal RB.
Double Precision NEGATE Register	BC	DNEG	R	The negative (2's complement) of the d.p. D0(RB,RB+1) is stored into (RA,RA+1). Status is set based on the result in (RA,RA+1). NOTE: The negative of zero is zero. : RA(RB) must be an even numbered register. : RA may equal RB.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Single Precision COMPARE	F0 F1 F2 F3 36 37	C CR CI CIM CISP CISN	D,DX R I,IX IM ISP ISN	The s.p. contents of the DO is compared to the contents of RA. Then status is set based on whether the contents of RA is less than, equal to, or greater than the DO. The contents of RA is unchanged.
Double Precision COMPARE	F4 F5 F6	DC DCR DCI	D,DX R I,IX	The d.p. is compared to the d.p. contents of (RA,RA+1). Status is set based on whether the contents of (RA,RA+1) is less than, equal to, or greater than the DO. The contents of (RA,RA+1) are unchanged. NOTE: RA(RB) must be an even numbered register.
Floating COMPARE	F7 F8 F9	FC FCR FCI	D,DX R I,IX	The f.p. contents of (RA,RA+1) is compared to the f.p. DO. Status is set based on whether the contents of (RA,RA+1) is less than, equal to, or greater than the DO. The contents of (RA,RA+1) are unchanged. NOTE: RA(RB) must be an even numbered register. : Operands must be normalized or f.p. zeros.
Floating Point ADD	A7 A8 A9 20 - 23	FA FAR FAI FAB	D,DX R I,IX B	The f.p. DO is floating point added to the f.p. contents of (RA,RA+1). The result, a normalized f.p. sum or f.p. zero, is stored in (RA,RA+1). Status is set based upon the result in (RA,RA+1) and exponent overflow or underflow. NOTE: RA(RB) must be an even numbered register. : Operands must be normalized or f.p. zeros.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Floating Point SUBTRACT	B7 B8 B9 24 - 27	FS FSR FSI FSB	D,DX R I,IX B	The f.p. D0 is floating point subtracted from the f.p. contents of (RA,RA+1). The result, a normalized f.p. difference or f.p. zero, is stored in (RA,RA+1). Status is set based upon the result in (RA,RA+1) and exponent overflow or underflow. NOTE: RA(RB) must be an even numbered register. : Operands must be normalized or f.p. zeros.
Floating Point MULTIPLY	C7 C8 C9 28 - 2B	FM FMR FMI FMB	D,DX R I,IX B	The f.p. D0 is floating point multiplied by the f.p. contents of (RA,RA+1). The result, a normalized f.p. product or f.p. zero, is stored in (RA,RA+1). Status is set based on the result in (RA,RA+1) and exponent overflow or underflow. NOTE: RA(RB) must be an even numbered register. : Operands must be normalized or f.p. zeros.
Floating Point DIVIDE	D7 D8 D9 20 - 2F	FD FDR FDI FDB	D.DX R I,IX B	The f.p. D0 is floating point divided by the f.p. contents of (RA,RA+1). The result, a normalized f.p. quotient or f.p. zero, is stored in (RA,RA+1). Status is set based on the result in (RA,RA+1) and exponent overflow or underflow. NOTE: RA(RB) must be an even numbered register. : Operands must be normalized or f.p. zeros.
Floating Point ABSOLUTE Value of Register	AD	FABS	R	The normalized absolute value of the f.p. D0 is stored in (RA,RA+1). Status is set based on the result in (RA,RA+1) and exponent overflow. NOTE: RA(RB) must be an even numbered register. : Operands must be normalized or f.p. zeros. : Overflow occurs if the D0 = -1.0 X 2**127.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Floating Point NEGATE Register	BD	FNEG	R	The mantissa of the f.p. DO (RB,RB+1) is 2's complemented. The exponent remains the same. The result, the negative of the original number, is stored in (RA,RA+1). Exceptions to this are all powers of two. Negation of 0.5 × 2**n is -1.0 × 2**n-1 and negation of -1.0 × 2**n is 0.5 × 2**n+1. Status is set based on the result in (RA,RA+1) and overflow. Overflow occurs if the DO = -1.0 × 2**127. NOTE: RA(RB) must be an even numbered register. : RA may equal RB. : Operands must be normalized or f.p. zeros.
CONVERT Integer to Floating Point	8D	FLT	R	The s.p. integer DO (RB) is converted into a normalized floating point number and is stored in (RA,RA+1). Status is set based on the result in (RA,RA+1). NOTE: RA(RB) must be an even numbered register. : RA may equal RB. : An integer zero is converted into a f.p. zero.
CONVERT Floating Point to Integer	8C	FIX	R	The f.p. DO (RB,RB+1) is converted into a s.p. integer value and is stored in RA. Status is set based on the result in RA. Overflow is set if the magnitude of the DO is too large to be converted into a 16-bit integer. NOTE: RB must be an even numbered register. : The DO must be normalized or f.p. zero.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Logical AND	E4 E5 E6 E7	AND ANDR ANDI ANDM	D,DX R I,IX IM	The DO is bit-by-bit AND'ed with the contents of RA. The result is stored in RA. Status is set based on the result in RA.
Inclusive Logical OR	E0 E1 E2 E3	OR ORR ORI ORIM	D,DX R I,IX IM	The DO is bit-by-bit inclusively OR'ed with the contents of RA. The result is stored in RA. Status is set based on the result in RA.
Exclusive Logical OR	E8 E9 EA EB	XOR XORR XORI XORM	D,DX R I,IX IM	The DO is bit-by-bit exclusively OR'ed with the contents of RA. The result is stored in RA. Status is set based on the result in RA.
Logical NAND	EC ED EE EF	N NR NI NIM	D,DX R I,IX IM	The DO is bit-by-bit logically NAND'ed with the contents of RA. The result is stored in RA. Status is set based on the result in RA. NOTE: The logical NOT of a register can be attained with a NR instruction and RA = RB.
SHIFT Left Logical	60	SLL	R	The contents of the DA (RB) are shifted left logically N positions. N is an integer ($1 \leq N \leq 16$) which is obtained by adding one to the value of the RA field of the instruction. Status is set based on the result in RB. NOTE: Logical Left shift enters zeros into LSB of RB.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
SHIFT Right Logical	61	SRL	R	The contents of the DA(RB) are shifted right logically N positions. N is an integer ($1 \leq N \leq 16$) which is obtained by adding one to the value of the RA field of the instruction. The result is saved in RB. Status is set based on the result in RB. NOTE: Logical right shift enters zeros into the MSB of RB.
SHIFT Right Arithmetic	62	SRA	R	The contents of the DA (RB) are shifted right arithmetically N positions. N is an integer ($1 \leq N \leq 16$) which is obtained by adding one to the value of the RA field of the instruction. Status is set based on the result in RB. NOTE: Arithmetic right shift enters the sign bit into the MSB of RB.
SHIFT Left Cyclic	63	SLC	R	The contents of the DA (RB) are shifted left cyclically N positions. N is an integer ($1 \leq N \leq 16$) which is obtained by adding one to the value of the RA field of the instruction. Status is set based on the result in RB. NOTE: Cyclic left shift enters the bit shifted out of the sign bit into the LSB of RB.
SHIFT Right Cyclic	64	SRC	R	The contents of the DA (RB) are shifted right cyclically N positions. N is an integer ($1 \leq N \leq 16$) which is obtained by adding one to the value of the RA field of the instruction. Status is set based on the result in RB. NOTE: Cyclic right shift enters the bit shifted out of the LSB into the MSB (sign bit) of RB.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Double SHIFT Logical Count in Register	6D	DSLR	R	The concatenated contents of (RA,RA+1) are shifted logically N positions, where N is the contents of RB. If N is positive, the shift direction is left; if N is negative, the shift direction is right. The final value in RB is zero. Status is set based on the result in RA. If |N| > 32, the illegal OC flag is set and no shifting takes place (i.e. NOP). NOTE: RA must be an even numbered register. : RA should not equal RB.
Double SHIFT Arithmetic Count in Register	6E	DSAR	R	The concatenated contents of (RA,RA+1) are shifted logically N positions, where N is the contents of RB. If N is positive, the shift direction is left; if N is negative, the shift direction is right. The final value in RB is zero. Status is set based on the result in RA. If |N| > 32, the illegal OC flag is set and no shifting takes place (i.e. NOP). NOTE: RA must be an even numbered register. : RA should not equal RB.
Double SHIFT Cyclic Count in Register	6F	DSCR	R	The concatenated contents of (RA,RA+1) are shifted logically N positions, where N is the contents of RB. If N is positive, the shift direction is left; if N is negative, the shift direction is right. The final value in RB is zero. Status is set based on the result in RA. If |N| > 32, the illegal OC flag is set and no shifting takes place (i.e. NOP). NOTE: RA must be an even numbered register. : RA should not equal RB.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
SET BIT	50 51 52	SB SBR SBI	D,DX R I,IX	Bit number N of the DO is set to one. N is an integer ($0 \leq N \leq 15$) which is obtained from the value of the 4-bit RA field of the instruction. NOTE: The MSB is designated bit number zero and the LSB is designated bit number 15.
RESET BIT	53 54 55	RB RBR RBI	D,DX R I,IX	Bit number N of the DO is set to zero. N is an integer ($0 \leq N \leq 15$) which is obtained from the value of the 4-bit RA field of the instruction.
TEST BIT	56 57 58	TB TBR TBI	D,DX R I,IX	Bit number N of the DO is tested. Then status is set to non-zero if bit number N contains one. Otherwise status is set to zero. N is an integer ($0 \leq N \leq 15$) which is obtained from the value of the RA field of the instruction.
SET Variable Bit in Register	5A	SVBR	R	Bit number N of register RB is set to one where N is an integer ($0 \leq N \leq 15$) which is obtained from the least significant four bits of the contents of RA. The most significant 12-bits of RA have no effect on the operation.
RESET Variable Bit in Register	5C	RVBR	R	Bit number N of register RB is set to zero where N is an integer ($0 \leq N \leq 15$) which is obtained from the least significant four bits of the contents of RA. The most significant 12-bits of RA have no effect on the operation.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
TEST Variable Bit in Register	5E	TVBR	R	Bit number N ($0 \leq N \leq 15$) of register RB is tested. Status is then set to non-zero if bit number N of RB is one. Otherwise, status is set to zero. N is obtained from the least significant four bits of the contents of RA.
Single Precision LOAD	80 81 82 83 00 - 03 38 39	L LR LI LIM LB LISP LISN	D,DX R I,IX IM,IMX B ISP ISN	The s.p. DO is loaded into register RA. Status is set based on the result in RA.
Double Precision LOAD	84 85 86 04 - 07	DL DLR DLI DLB	D,DX R I,IX B	The d.p. DO is loaded into (RA,RA+1) such that the MSH of the DO is in RA. Status is set based on the result in (RA,RA+1). NOTE: RA(RB) must be an even numbered register.
LOAD From Upper Byte	87 7C	LUB LUBI	D,DX I,IX	The MSH (upper byte) of the DO is loaded into the LSH (lower byte) of register RA. The MSH (upper byte) of RA is set to zero. Status is set based on the result in RA.
LOAD From Lower Byte	88 7D	LLB LLBI	D,DX I,IX	The LSH (lower byte) of the DO is loaded into the LSH (lower byte) of register RA. The MSH (upper byte) of RA is set to zero. Status is set based on the result in RA.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
LOAD STATUS	8E	LDST	D,DX	The contents of the Derived Address, DA, and DA+1, and DA+2 are loaded into the Interrupt Mask register, Status Word Register and Instruction Counter, respectively.
RETURN From Interrupt	8B	RFI	R	The Interrupt Mask Register, Status Word Register, and Instruction Counter are sequentially loaded from a memory stack pointed to by register RA. The value in RA is decremented by 3.
LOAD Multiple Registers	8F	LM	D,DX	The contents of the DA are loaded into register R0, then the contents of DA+1 are loaded into R1,..., finally, the contents of DA+N are loaded into register RN. Effectively, this instruction allows the transfer of (N+1) words from memory to the register file. N is an integer ($0 \leq N \leq 15$) which is obtained from the 4-bit RA field of the instruction.
Single Precision STORE	90 92 08 - 0B	ST STI STB	D,DX I,IX B	The contents of register RA are stored into the Derived Address, DA.
Double Precision STORE	94 96 0C - 0F	DST DSTI DSTB	D,DX I,IX B	The contents of registers RA and RA+1 are stored at the Derived Address, DA, and DA+1, respectively. NOTE: RA must be an even numbered register.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
STORE a Non-Negative Constant	91 93	STC STCI	D,DX I,IX	The constant N ($0 < N \leq 15$) is stored at the Derived Address (DA). The value N is obtained from the 4-bit RA field of the instruction.
STORE Register Through Mask	95	SRM	D,DX	The contents of register RA is stored into the DA through the mask in register RA+1. For each position in the mask that is one, the corresponding bit of RA is stored into the corresponding bit of the DA. For each position in the mask that is zero, no change is made to the corresponding bit in the DA. NOTE: Register RA must be an even numbered register.
STORE into Upper Byte	97 7E	STUB SUBI	D,DX I,IX	The LSH (lower byte) of RA is stored into the MSH (upper byte) of the Derived Address (DA). The LSH of the DA is unchanged.
STORE into Lower Byte	98 7F	STLB SLBI	D,DX I,IX	The LSH (lower byte) of RA is stored into the LSH (lower byte) of the Derived Address (DA). The MSH of the DA is unchanged.
MOVE Multiple Words, Memory-to-Memory	9C 9D	MOV MOVI	D,DX I,IX	This instruction allows a memory-to-memory transfer of N words where N is an integer between zero and ((2**16)-1) and is obtained from the contents of register RA+1. The contents of the DA is the first word to be transferred and the contents of RA represents the address of where the first word is to be transferred. The final value left in RA is the last stored address plus one, RA+1 has a final value of zero. NOTE: RA must be an even numbered register.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
STORE Multiple Registers	9F	STM	D,DX	This instruction transfers N+1 words from the register file to memory. The contents of register R0 is stored into the DA; then the contents of R1 is stored into DA+1; ··· ; finally,the contents of RN is stored into DA+N where N is an integer ($0 < N < 15$). The value of N is obtained from the 4-bit RA field of the instruction.
PUSH Multiple Registers onto Stack	89	PSHM	R	The contents of registers RB through RB+N are pushed onto a stack in memory using R15 as the stack pointer. After the contents of the last register, RB+N, is pushed onto the stack, R15 is incremented by N+1. The number N is an integer which is obtained from the 4-bit RA field of the instruction. It is the responsibility of the programmer to ensure that the value of RB+N is less than or equal to 15. If RB+N is greater than 15, registers RB through R15, and then R0 through RB+N-16 are pushed onto the stack.
POP Multiple Registers off Stack	99	POPM	R	Registers RB+N through RB are loaded sequentially from the memory stack pointed to by register R15. Once the last register, RB, is loaded, the contents of R15 are decremented by N+1. If R15 is included in the transfer, it is effectively ignored. The number N is an integer which is obtained from the 4-bit RA field of the instruction. It is the programmer's responsibility to ensure that RB+N is less than or equal to 15. If RB+N is greater than 15, registers RB+N-16 through R0, and R15 through RB are popped from the stack.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Stack IC and JUMP to Subroutine	7B	SJS	D,DX	Register RA is the stack pointer. The contents of RA are incremented by one and the address of the instruction following the SJS instruction is stored into the memory location pointed to by RA. Program control is then transferred to the instruction at the DA.
Unstack IC and RETURN from Subroutine	7A	URTS	R	Register RA is the stack pointer. The contents of the memory location pointed to by RA are loaded into the instruction counter (IC). The value in RA is then decremented by one.
NO OPERATION	FF	NOP	R	No operation is performed.
Break Point	77	BPT	R	This instruction is typically used for halting the processor during maintenance and diagnostic procedures when the console is connected to the system. If the console is not connected the instruction is treated as a NOP. Restarting the processor after a BPT can only be effected by : (1) the maintenance console; or (2) the power on sequence; or (3) an interrupt.
EXECUTE	76	EX	D,DX	The instruction located at the DA is executed. If that instruction causes a jump, then execution continues from the jump location. If the instruction did not cause a jump, then the next sequential instruction after this EXECUTE instruction is executed next.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Double SHIFT Left Logical	65	DSLL	R	The concatenated contents of the DA (RB,RB+1) are shifted left logically N positions. N is an integer ($1 \leq N < 16$) which is obtained by adding one to the value of the RA field of the instruction. Status is set based upon the result in (RB,RB+1). NOTE: RB must be an even numbered register.
Double SHIFT Right Logical	66	DSRL	R	The concatenated contents of the DA (RB,RB+1) are shifted right logically N positions. N is an integer ($1 \leq N < 16$) which is obtained by adding one to the value of the RA field of the instruction. Status is set based on the result in (RB,RB+1). NOTE: RB must be an even numbered register.
Double SHIFT Right Cyclic	67	DSRC	R	The concatenated contents of the DA (RB,RB+1) are shifted right cyclically N positions. N is an integer ($1 \leq N \leq 16$) which is obtained by adding one to the value of the RA field of the instruction. Status is set based on the result in (RB,RB+1). NOTE: RB must be an even numbered register.
Double SHIFT Left Cyclic	68	DSLC	R	The concatenated contents of the DA (RB,RB+1) are shifted left cyclically N positions. N is an integer ($1 \leq N < 16$) which is obtained by adding one to the value of the RA field of the instruction. Status is set based on the result in (RB,RB+1). NOTE: RB must be an even numbered register.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Double SHIFT Right Arithmetic	69	DSRA	R	The concatenated contents of the DA (RB,RB+1) are shifted right arithmetically N positions. N is an integer ($1 < N < 16$) which is obtained by adding one to the value of the RA field of the instruction. Status is set based upon the result in (RB,RB+1). NOTE: RB must be an even numbered register.
SHIFT Logical Count in Register	6A	SLR	R	The contents of RA are shifted logically N positions, where N is the contents of RB. If N is positive, the shift direction is left; if N is negative, the shift direction is right. Status is set based on the result in RA. The final value in RB is zero. If $\|N\| > 16$, the illegal OC flag is set and no shifting takes place (i.e. NOP).
SHIFT Arithmetic Count in Register	6B	SAR	R	The contents of RA are shifted arithmetically N positions, where N is the contents of RB. If N is positive, the shift direction is left; if N is negative, the shift direction is right. Status is set based on the result in RA. The final value in RB is zero. If $\|N\| > 16$, the illegal OC flag is set and no shifting takes place (i.e. NOP).
SHIFT Cyclic Count in Register	6C	SCR	R	The contents of RA are shifted cyclically N positions, where N is the contents of RB. If N is positive, the shift direction is left; if N is negative, the shift direction is right. Status is set based on the result in RA. The final value in RB is zero. If $\|N\| > 16$, the illegal OC flag is set and no shifting takes place (i.e. NOP).

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
Jump Back and Count	74	JBC	D,DX	If the contents of register RA is zero, then the next sequential instruction is executed. If the contents of RA is non-zero, then RA is decremented by one and a jump to the Derived Address (DA) occurs. A zero value in RA remains zero.
BRANCH if Equal to Zero	30	BEZ	ICR	A program branch is made to the Derived Address (DA) if status indicated that the previous result which set status was equal to zero. Otherwise, the next sequential instruction is executed.
BRANCH if Less Than Zero	31	BLT	ICR	A program branch is made to the Derived Address (DA) if the status indicated that the previous result which set status was less than zero. Otherwise the next sequential instruction is executed.
BRANCH if Less Than or Equal to Zero	32	BLE	ICR	A program branch is made to the Derived Address (DA) if status indicates that the previous result which set status was less than or equal to zero. Otherwise, the next sequential instruction is executed.
BRANCH if Greater Than Zero	33	BGT	ICR	A program branch is made to the Derived Address (DA) if status indicates that the previous result which set status was greater than zero. Otherwise, the next sequential instruction is executed.

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
BRANCH if NOT Equal to Zero	34	BNZ	ICR	A program branch is made to the Derived Address (DA) if status indicates that the previous result which set status was not equal to zero. Otherwise, the next sequential instruction is executed.
BRANCH if Greater Than or Equal to Zero	35	BGE	ICR	A program branch is made to the Derived Address (DA) if status indicates that the previous result which set status was greater than or equal to zero. Otherwise, the next sequential instruction is executed.
BRANCH Always	75	BR	ICR	A program branch is made to the Derived Address (DA).
JUMP to Subroutine	72	JS	D,DX	The value of the Instruction Counter (IC) is incremented by one and stored into register RA. Then the IC is set to the Derived Address (DA), thus effecting the jump. Return from subroutine is accomplished by an indexed unconditional jump to location zero using RA as the index register.
Console Input/Output	79	CIO	IM	This instruction is reserved for input and output of peripherals via a console unit for the microcomputer. If there is no console connected to the computer, the CIO instruction is treated as a NOP. (The details of this instruction have not been defined. The instruction is therefore treated as a NOP.)

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
INPUT	8A	IN	IM,IMX	The Input instruction transfers data from an I/O device to register RA. The value of the DO can be viewed as an opcode extension (OXC) which specifies the operation to be performed or the device to be addressed. The mnemonic and OXC for some pre-assigned input operations are listed below. MNEMONIC — FUNCTION — OXC VALUE Hex ITA — From Timer A to (RA) — C001 ITB — From Timer B to (RA) — C002 RSW — From Status Word to (RA) — C003 ID — From Discrete Lines to (RA) — C009 RIC — From Interrupt Mask to (RA) — C00B PI — From PIO Channel to (RA) — 8000 + PIO ADDR
OUTPUT	9A	OUT	IM,IMX	The Output instruction transfers data from register RA to an I/O device. The value of the DO can be viewed as an opcode extension (OXC) which specified the operation to be performed or the device to be addressed. The mnemonic and OXC for some pre-assigned output operations are listed below. MNEMONIC — FUNCTION — OXC VALUE Hex GO — Trigger GO/NO GO Indicator — 4000 OTA — To Timer A From (RA) — 4001 OTB — To Timer B From (RA) — 4002 DMAE — DMA Enable — 4005 DMAD — DMA Disable — 4006 DSBL — Interrupt Disable — 4008 OD — To Discrete Lines From (RA) — 4009 CLIR — Clear Pending Interrupts — 400A SIC — Set Interrupt Mask From (RA) — 400B ENBL — Interrupt Enable — 400C PO — To PIO Channel From (RA) — 0000 + PIO ADDR

Table A4 (continued)

INSTRUCTION NAME	OP CODE	MNEMONIC	ADDRESS MODE	DESCRIPTION
JUMP on Condition	70 71	JC JCI	D,DX I,IX	A program jump to the DA occurs if a logical one results from the following operation. 1. The 4-bit condition status (CS) output by the Status Register is Bit-by-Bit AND'ed with the 4-bit RA field of the instruction. 2. The resulting 4-bits are OR'ed together. Otherwise, the next sequential instruction is executed.

Appendix B

Microinstruction Word Description

Figure B1 shows the format of the microinstruction word. Tables B1 through B15 describe the microcode and mnemonic symbols associated with each microinstruction field. These symbols are used in Appendix C to describe the emulation microprogram.

Bit Steering

The Interrupt Control, Direct Data Short, Direct Data Long, and Direct Address fields of the microinstruction word are overlapped. Bit steering for these fields is accomplished in the following manner:

1. Microinstruction Bits I_{47} - I_{50} are interpreted as the Interrupt Control field if the "INTE" microinstruction is issued by Command field.

2. Microinstruction Bits I_{51} - I_{58} are interpreted as the Direct Data Short (8-bits) field if the "PLJK" microinstruction is issued by JK MUX field.

3. Microinstruction Bits I_{43} - I_{58} are interpreted as the Direct Data Long (16-bits) field if the "uDL" microinstruction is issued by Register Out field.

4. Microinstruction Bits I_{47} - I_{58} are interpreted as the Direct Address field depending upon the microinstruction issued by the Next Address field.

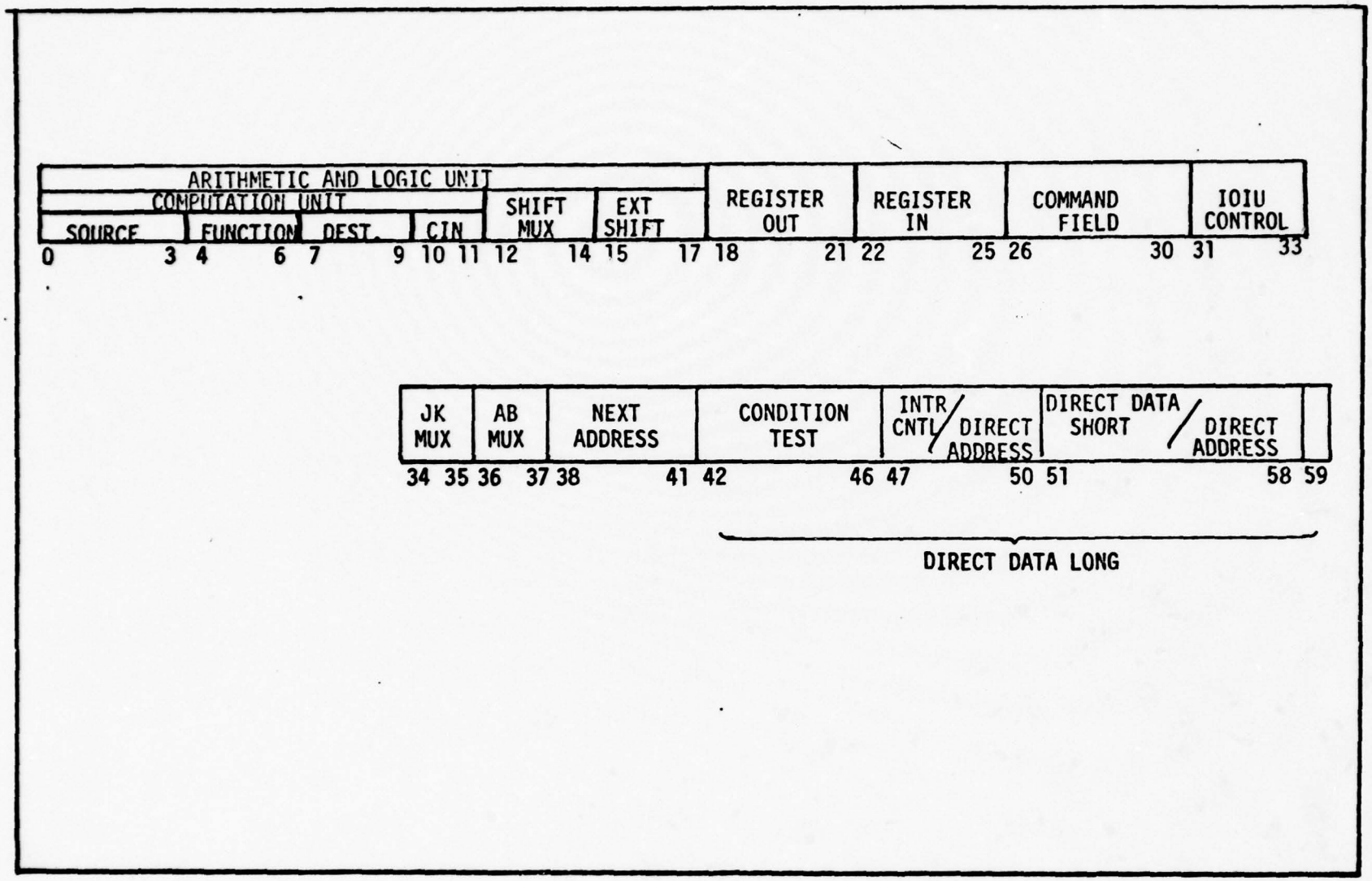


Fig. B1. Microinstruction Word Format

Abbreviations

The following abbreviations are used in this appendix to describe the microinstruction word.

Notation	Name	Meaning
$X \rightarrow Y$	Specification	New value of variable Y is equal to current value of variable X.
X + Y	Addition	Variables X and Y are (2's complemented) ADD'ed
X / Y	Division	Variable X is Divided by Y.
$X \neq Y$	Not Equal	Variable X is not equal to variable Y.
$X \vee Y$	Logical OR	Variable X is bit by bit logically OR'ed with variable Y.
$X \veebar Y$	Exclusive OR	Variable X is bit by bit exclusively OR'ed with variable Y.
$X \wedge Y$	Logical AND	Variable X is bit by bit loically AND'ed with variable Y.
$\overline{X}$	NOT	Variable X is complemented.
(X,Y)	Concatenate	Variables X and Y are concatenated in such a way that X is the most significant half.
RSHT(X)	Right Shift	Variable X is shifted right one bit and the vacated bit is filled from Right MUX (Table B5).
LSHT(X)	Left Shift	Variable X is shifted left one bit and the vacated bit is filled from Left MUX (Table B5).
*	Default Instruction	Microprogram documentation in Appendix C refers to a default microinstruction for each microinstruction field. These default microinstructions are identified with an asterisk.

Table B1

ALU Source Field

SYMBOL	MICRO CODE				SOURCE	
	I0	I1	I2	I3	S	R
AQ	0	0	0	0	RA	Q
AB	0	0	0	1	RA	RB
0Q	0	0	1	0	0	Q
0B	0	0	1	1	0	RB
0A	0	1	0	0	0	RA
DA	0	1	0	1	D	RA
DQ	0	1	1	0	D	Q
* D0	0	1	1	1	D	0
AQ+	1	0	0	0	RA+1	Q
AB+	1	0	0	1	RA+1	RB+1
0Q	1	0	1	0	0	Q
0B+	1	0	1	1	0	RB+1
0A+	1	1	0	0	0	RA+1
DA+	1	1	0	1	D	RA+1
DQ	1	1	1	0	D	Q
D0	1	1	1	1	D	0

Table B2

ALU Function Field

SYMBOL	MICRO CODE			DESCRIPTION
	I4	I5	I6	
ADD	0	0	0	R + S + CIN → F
S-R	0	0	1	S + $\overline{R}$ + CIN → F
R-S	0	1	0	R + $\overline{S}$ + CIN → F
* OR	0	1	1	R V S → F
AND	1	0	0	$\overline{R}$ ∧ S → F
NRAS	1	0	1	$\overline{R}$ ∧ S → F
XOR	1	1	0	R ⊻ S → F
XNOR	1	1	1	$\overline{R \veebar S}$ → F

Table B3
ALU Destination Field

SYMBOL	MICRO CODE			RAM FUNCTION		Q FUNCTION		OUTPUT
	I7	I8	I9	SHIFT	LOAD	SHIFT	LOAD	
QOF	0	0	0	- -	NONE	NONE	F → Q	F → OBUS
* OF	0	0	1	- -	NONE	-	NONE	F → OBUS
BOA	0	1	0	NONE	F → (RB)	-	NONE	(RA)→ OBUS
BOF	0	1	1	NONE	F → (RB)	-	NONE	F → OBUS
RBQ	1	0	0	RIGHT	F/2 → (RB)	RIGHT	Q/2 → Q	F → OBUS
RB	1	0	1	RIGHT	F/2 → (RB)	-	NONE	F → OBUS
LBQ	1	1	0	LEFT	2F → (RB)	LEFT	2Q → Q	F → OBUS
LB	1	1	1	LEFT	2F → (RB)	-	NONE	F → OBUS

Table B4
Carry-In Field

SYMBOL	MICRO CODE		DESCRIPTION
	I10	I11	
*CZRO .	0	0	0 → CIN
CONE	0	1	1 → CIN
CLNK	1	0	LINK → CIN
- -	1	1	SPARE

Table B5
Shift Multiplexer Field

SYMBOL	MICRO CODE I12	I13	I14	RIGHT MUX OUTPUT	LEFT MUX OUTPUT
*SZRO	0	0	0	0	0
SONE	0	0	1	1	1
SQO	0	1	0	RQ0	LQ0
SRO	0	1	1	RR0	LR0
SOBO	1	0	0	$OBUS_0$	$OBUS_0$
SYO	1	0	1	Z_{15}	Y_0
SWO	1	1	0	W_0	W_0
SLNK	1	1	1	LINK	LINK

Table B6
External Shift Field

SYMBOL	MICRO CODE I15	I16	I17	DESCRIPTION
* - -	0	0	0	NO ACTION
- -	0	0	1	NO ACTION
LY	0	1	0	LSHT(Y,Z)
RY	0	1	1	RSHT(Y,Z)
LW	1	0	0	LSHT(W,X)
RW	1	0	1	RSHT(W,X)
LWY	1	1	0	LSHT(W,X); LSHT(Y,Z)
RWY	1	1	1	RSHT(W,X); RSHT(Y,Z)

Table B7
Register Out Field

SYMBOL	MICRO CODE I18	I19	I20	I21	DESCRIPTION
* --	0	0	0	0	NO ACTION
OUTW	0	0	0	1	W → DBUS
OUTX	0	0	1	0	X → DBUS
OUTY	0	0	1	1	Y → DBUS
OUTZ	0	1	0	0	Z → DBUS
OUTG	0	1	0	1	G → DBUS
OUTH	0	1	1	0	H → DBUS
OMDR	0	1	1	1	MDR → DBUS
OJK	1	0	0	0	JK → DBUS
OUTS	1	0	0	1	STATUS → DBUS
ATOD	1	0	1	0	ABUS → DBUS
OTOD	1	0	1	1	OBUS → DBUS
uDL	1	1	0	0	I43-58 → DBUS
TAO	1	1	0	1	TIMER A → DBUS
TBO	1	1	1	0	TIMER B → DBUS
BMO	1	1	1	1	Bit Mask Output→ DBUS

Table B8
Register In Field

SYMBOL	MICRO CODE I22	I23	I24	I25	DESCRIPTION
* --	0	0	0	0	NO ACTION
INIC	0	0	0	1	OBUS → IC
INEA	0	0	1	0	OBUS → EAR
INIR	0	0	1	1	DBUS → IR
INW	0	1	0	0	OBUS → W
INX	0	1	0	1	OBUS → X
INY	0	1	1	0	OBUS → Y
INZ	0	1	1	1	OBUS → Z
ING	1	0	0	0	OBUS → G
INH	1	0	0	1	OBUS → H
INS	1	0	1	0	DBUS → STATUS
INN	1	0	1	1	DBUS → N
- -	1	1	0	0	SPARE
- -	1	1	0	1	SPARE
- -	1	1	1	0	SPARE
- -	1	1	1	1	SPARE

Table B9
Command Field

SYMBOL	MICRO CODE I26	I27	I28	I29	I30	DESCRIPTION
* - -	0	0	0	0	0	NO ACTION
ISGN	0	0	0	0	1	$\overline{\text{SIGN flip flop}}$ → SIGN flip flop
LDZO	0	0	0	1	0	FZERO → DZRO flip flop
- -	0	0	0	1	1	SPARE
LST	0	0	1	0	0	Latch Status
LSTU	0	0	1	0	1	Latch Status w Underflow
LSTV	0	0	1	1	0	Latch Status w Overflow
LSTC	0	0	1	1	1	Latch Status w OC Error
- -	0	1	0	0	0	SPARE
MAPX	0	1	0	0	1	Enable I/O Mapping PROM
INTE	0	1	0	1	0	Interrupt Instruction Enable
- -	0	1	0	1	1	SPARE
OEAR	0	1	1	0	0	EAR → ABUS
OIC	0	1	1	0	1	IC → ABUS
OTOA	0	1	1	1	0	OBUS → ABUS
- -	0	1	1	1	1	SPARE
- -	1	0	0	0	0	SPARE
- -	1	0	0	0	1	SPARE
TAIN	1	0	0	1	0	DBUS → TIMER A
TBIN	1	0	0	1	1	DBUS → TIMER B
DECN	1	0	1	0	0	Decrement N Counter
- -	1	0	1	0	1	SPARE
SEX	1	0	1	1	0	Set EX flip flop
CEX	1	0	1	1	1	Clear EX flip flop
DMAD	1	1	0	0	0	DMA Channel Disable
DMAE	1	1	0	0	1	DMA Channel Enable
TGNG	1	1	0	1	0	Trigger GO/NO GO Indicator
BPNT	1	1	0	1	1	Break Point Command
CLDZ	1	1	1	0	0	Clear DZRO flip flop
CLWX	1	1	1	0	1	Clear Registers W and X
CLYZ	1	1	1	1	0	Clear Registers Y and Z
PCLR	1	1	1	1	1	Clear: W, X, Y, Z, G, H, SIGN, & DZRO

Table B10
Input/Output Interface Unit Field

SYMBOL	MICRO CODE I31	I32	I33	DESCRIPTION
*- -	0	0	0	NO ACTION
- -	0	0	1	SPARE
DWRQ	0	1	0	Discrete I/O Write Request
DRRQ	0	1	1	Discrete I/O Read Request
PWRQ	1	0	0	PIO Write Request
PRRQ	1	0	1	PIO Read Reqeust
MWRQ	1	1	0	Memory Write Request
MRRQ	1	1	1	Memory Read Request

Table B11
JK Multiplexer Field

SYMBOL	MICRO CODE I34	I35	DESCRIPTION
*IRJK	0	0	$IR_{8-11} \rightarrow J$; $IR_{12-15} \rightarrow K$
IRKJ	0	1	$IR_{12-15} \rightarrow J$; $IR_{8-11} \rightarrow K$
PLJK	1	0	$I_{51-54} \rightarrow J$; $I_{55-58} \rightarrow K$
DBJK	1	1	$DBUS_{8-11} \rightarrow J$; $DBUS_{12-15} \rightarrow K$

Table B12
AB Latch Field

SYMBOL	MICRO CODE I36	MICRO CODE I37	DESCRIPTION
*HOLD	0	0	HOLD A, B
INA	0	1	J → A: HOLD B
INB	1	0	K → B; HOLD A
INAB	1	1	J → A; K → B

Table B13
Next Address Field

SYMBOL	MICRO CODE I38	I39	I40	I41	DESCRIPTION
JZ	0	0	0	0	Jump Zero
CJS	0	0	0	1	Cond. Jump SBR PL
JMAP	0	0	1	0	Jump MAP Address
CJP	0	0	1	1	Cond. Jump PL
PUSH	0	1	0	0	Push Stack, Con. Load R
JSRP	0	1	0	1	Jump SBR R/PL
CJV	0	1	1	0	Cond. Jump Vector
JRP	0	1	1	1	Jump R/PL
RFCT	1	0	0	0	Repeat Loop; N ≠ 0
RPCT	1	0	0	1	Repeat PL; N ≠ 0
CRTN	1	0	1	0	Cond. Return
CJPP	1	0	1	1	Cond. Jump PL & POP
LDCT	1	1	0	0	Load CNTR & Continue
LOOP	1	1	0	1	Test End of Loop
*CONT	1	1	1	0	Continue
TWB	1	1	1	1	Three Way Branch

Table B14
Condition Test Field

SYMBOL	MICRO CODE I42	I43	I44	I45	I46	TEST DESCRIPTION (Test Satisfied If)
* NO	0	0	0	0	0	Test Never Satisfied
WO = 1	0	0	0	0	1	$W_0 = 1$
NRM	0	0	0	1	0	$W_0 \neq W_1 = 1$
W1 = 1	0	0	0	1	1	$W_1 = 1$
X15	0	0	1	0	0	$X_{15} = 1$
OVR	0	0	1	0	1	OVERFLOW Flag = 1
FO = 1	0	0	1	1	0	FO Flag = 1
F = 0	0	0	1	1	1	FZERO Flag = 1
GO = 1	0	1	0	0	0	GO Flag = 1
INT	0	1	0	0	1	INTERRUPT Flag = 1
LNK	0	1	0	1	0	LINK flip flop = 1
JOC	0	1	0	1	1	JOC Flag = 1
SGN	0	1	1	0	0	SIGN flip flop = 1
FLG	0	1	1	0	1	INDEX Flag = 1
N = 0	0	1	1	1	0	N COUNTER = 0
EX = 1	0	1	1	1	1	EX flip flop = 1
YES	1	0	0	0	0	Test Always Satisfied
WO = 0	1	0	0	0	1	$W_0 = 0$
≠ NRM	1	0	0	1	0	$W_0 \neq W_1 = 0$
W1 = 0	1	0	0	1	1	$W_1 = 0$
≠ X15	1	0	1	0	0	$X_{15} = 0$
≠ OVR	1	0	1	0	1	OVERFLOW Flag = 0
FO = 0	1	0	1	1	0	FO Flag = 0
F = 0	1	0	1	1	1	FZERO Flag = 0
GO = 0	1	1	0	0	0	GO Flag = 0
≠ INT	1	1	0	0	1	INTERRUPT Flag = 0
≠ LNK	1	1	0	1	0	LINK flip flop = 0
≠ JOC	1	1	0	1	1	JOC Flag = 0
≠ SGN	1	1	1	0	0	SIGN flip flop = 0
≠ FLG	1	1	1	0	1	INDEX Flag = 0
N ≠ 0	1	1	1	1	0	N COUNTER ≠ 0
EX = 0	1	1	1	1	1	EX flip flop = 0

Table B15
Interrupt Control Field

SYMBOL	MICRO CODE				DESCRIPTION
	I47	I48	I49	I50	
*IMCL	0	0	0	0	Interrupt Unit Master Clear
CAI	0	0	0	1	Clear all Interrupts
MBC	0	0	1	0	Clear Interrupts from BUS
IMC	0	0	1	1	Clear Intertupt from Mask
CLV	0	1	0	0	Clear Interrupt, Last Vector Read
RDV	0	1	0	1	Read Vector
RSR	0	1	1	0	Read Status Register
RMR	0	1	1	1	Read Mask Register
SMR	1	0	0	0	Set Mask Register
LSR	1	0	0	1	Load Status Register
BCM	1	0	1	0	Bit Clear Mask Register
BSM	1	0	1	1	Bit Set Mask Register
CMR	1	1	0	0	Clear Mask Register
DIR	1	1	0	1	Disable Interrupt Requests
LMR	1	1	1	0	Load Mask Register
EIR	1	1	1	1	Enable Interrupt Requests

Appendix C

Emulation Microprogram

This appendix presents the emulation microprogram which was described in Chapter VII. The mnemonic symbols used in this appendix were defined in Appendix B.

Table C1
Emulation Microprogram

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
PWRU+ 0	DO	OR	FO	*	*	*	OJK	ING	*	*	PLJK	*	*	*	*	uDS(OF)
+ 1	DO	AND	BOF	*	*	*	OUTG	*	*	*	DBJK	INB	CJP	F=0	-	ADDR(PWRU+3)
+ 2	DO	S-R	FO	CZRO	*	*	OUTG	ING	*	*	*	*	CJP	YES	-	ADDR(PWRU+1)
+ 3	DO	AND	FO	*	*	*	OTOD	INS	PCLR	*	*	*	*	*	*	
+ 4	DO	AND	FO	*	*	*	OTOD	INIC	INTE	*	DBJK	INAB	PUSH	NO	IMCL	
+ 5	OB	S-R	FO	CZRO	*	*	OTOD	INIR	CEX	*	*	*	LOOP	GO=1	*	
+ 6	DO	AND	FO	*	*	*	*	INEA	OIC	MRRQ	*	*	CJP	YES	-	ADDR(IF+0)
IF+ 0	*	*	*	*	*	*	OMDR	INIR	PLCR	*	*	*	CJP	GO=0	-	ADDR(IF+0)
+ 1	DO	AND	FO	CONE	*	*	ATOD	INIC	OIC	*	*	*	JMAP	YES		
INTR+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJP	EX=1	-	ADDR(INTR+26)
+ 1	*	*	*	*	*	*	*	INEA	INTE	*	*	*	PUSH	NO	RDV	
+ 2	*	*	*	*	*	*	*	INX	INTE	*	*	*	LOOP	GO=1	RMR	
+ 3	DO	OR	OF	*	*	*	ATOD	INW	OEAR	MRRQ	PLJK	INB	*	*	*	uDS(OF)
+ 4	DO	OR	QOF	*	*	*	OMDR	*	*	*	*	*	CJP	GO=0	-	ADDR(INTR+4)
+ 5	OQ	R-S	QOF	CZRO	*	*	*	*	*	*	*	*	*	*	*	
+ 6	OQ	ADD	QOF	CONE	*	*	OUTX	*	OTOA	MWRQ	*	*	PUSH	NO	*	
+ 7	*	*	*	*	*	*	OJK	*	INTE	*	PLJK	*	LOOP	GO=1	LSR	uDS(OF)
+ 8	OQ	R-S	QOF	CZRO	*	*	OUTS	*	OTOA	MWRQ	*	*	PUSH	NO	*	
+ 9	DO	OR	OF	*	*	*	*	INX	OIC	*	*	*	LOOP	GO=1	*	
+10	OQ	OR	OF	*	*	*	OUTX	*	OTOA	MWRQ	*	*	CJP	YES	-	ADDR(INTR+16)
+11	OB	R-S	BOF	CZRO	*	*	OUTX	*	OTOA	MWRQ	*	*	PUSH	NO	*	
+12	*	*	*	*	*	*	OJK	*	INTE	*	PLJK	*	LOOP	GO=1	LSR	uDS(OF)
+13	OB	R-S	BOF	CZRO	*	*	OUTS	*	OTOA	MWRQ	*	*	PUSH	NO	*	
+14	DO	OR	OF	*	*	*	*	INX	OIC	*	*	*	LOOP	GO=1	*	
+15	OB	R-S	BOF	CZRO	*	*	OUTX	*	OTOA	MWRQ	*	*	*	*	*	
+16	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(INTR+16)
+17	DO	ADD	OF	CONE	*	*	OUTW	*	OTOA	MRRQ	*	*	*	*	*	
+18	*	*	*	*	*	*	OMDR	INEA	*	*	*	*	CJP	GO=0	-	ADDR(INTR+18)
+19	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	PUSH	*	*	
+20	*	*	*	*	*	*	OMDR	*	INTE	*	*	*	LOOP	GO=1	SMk	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
INTR+21	DO	ADD	OF	CONE	*	*	ATOD	*	OEAR	MRRQ	*	*	*	*	*	
+22	*	*	*	*	*	*	OMDR	INX	*	*	*	*	CJP	GO=0	-	ADDR(INTR+22)
+23	*	*	*	*	*	*	OUTX	INS	OEAR	MRRQ	*	*	*	*	*	
+24	DO	OR	OF	*	*	*	OMDR	INIC	*	*	*	*	CJP	GO=0	-	ADDR(INTR+24)
+25	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(IF+0)
+26	DO	OR	OF	*	*	*	OUTH	INIC	CEX	*	*	*	CJP	GO=0	-	ADDR(INTR+26)
+27	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	-	-	ADDR(IF+0)
+28	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DAM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	*	*	*	
+ 1	OB	OR	QOF	*	*	*	*	*	*	*	IRJK	INAB	CJP	GO=0	-	ADDR(DAM+1)
+ 2	DO	OR	OF	*	*	*	OMDR	INEA	*	*	IRKJ	INB	CRTN	≠FLG	*	
+ 3	DQ	ADD	OF	CZRO	*	*	ATOD	INEA	OEAR	*	*	*	CRTN	YES	*	
IAM+ 0	DO	ADD	OF	CONE	*	*	*	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 2	DO	OR	OF	*	*	*	OMDR	INEA	*	*	*	*	CRTN	GO=1	*	
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(IAM+2)
ICR+ 0	DO	OR	OF	*	*	*	OJK	ING	*	*	IRJK	*	*	*	*	
+ 1	DO	OR	QOF	*	*	*	OUTG	*	*	*	*	*	*	*	*	
+ 2	DQ	ADD	OF	CZRO	*	*	ATOD	INEA	OIC	*	*	*	CRTN	YES	*	
IMAM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	*	*	*	
+ 1	DO	OR	OF	*	*	*	OMDR	INW	*	*	IRJK	INAB	CJP	GO=0	-	ADDR(IMAM+1)
+ 2	OB	OR	QOF	*	*	*	*	*	*	*	*	*	CJP	≠FLG	-	ADDR(IMAM+4)
+ 3	DQ	ADD	OF	CZRO	*	*	OUTW	INW	*	*	IRKJ	INB	CRTN	YES	*	
+ 4	*	*	*	*	*	*	*	*	*	*	IRKJ	INB	CRTN	YES	*	
ISP+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRJK	INA	*	-	-	uDL(000F)
+ 1	DO	ADD	QOF	*	*	*	OJK	*	*	*	IRJK	*	*	*	*	
+ 2	OQ	ADD	OF	CZRO	*	*	*	INW	*	*	IRKJ	INB	CRTN	YES	*	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
ISN+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRJK	INA	*	-	-	uDL(000F)
+ 1	DQ	AND	QOF	*	*	*	OJK	*	*	*	IRJK	*	*	*	*	
+ 2	OQ	S-R	OF	CZRO	*	*	*	INW	*	*	IRKJ	INB	CRTN	YES	*	
AR+ 0	OB	OR	OF	*	*	*	*	INW	OIC	MRRQ	IRJK	INAB	*	*	*	
+ 1	*	*	*	*	*	*	*	*	*	*	IRKJ	INB	*	*	*	
+ 2	DA	ADD	BOF	CZRO	*	*	OUTW	*	LST	*	*	*	CJP	≠INT	-	ADDR(IF+0)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(INTR)
AB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(22)
+ 3	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(AB4+3)
+ 4	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(AR+2)
AB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(AB4+2)
AB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(AB4+2)
AB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(70)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(AB4+2)
A+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(AB4+3)
AI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(AB4+3)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
AISP+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISP+0)
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(AR+2)
AISN+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISN+0)
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(AR+2)
AIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+0)
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(AR+1)
DAR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	OB+	OR	OF	*	*	*	*	INX	*	*	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	OIC	MRRQ	IRKJ	INB	*	*	*	
+ 3	DA+	ADD	BOF	CZRO	*	*	OUTX	*	LDZO	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	DA	ADD	BOF	CLNK	*	*	OUTW	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DA+ 0	DO	ADD	OF	CONE	*	*	*	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(DA+2)
+ 3	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	*	ADDR(DA+4)
+ 4	*	*	*	*	*	*	OMDR	INX	*	*	*	*	JRP	GO=0	-	ADDR(DAR+2)
DAI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	*	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(DA+2)
SR+ 0	OB	OR	OF	*	*	*	*	INW	OIC	MRRQ	IRJK	INAB	*	*	*	
+ 1	*	*	*	*	*	*	*	*	*	*	IRKJ	INB	*	*	*	
+ 2	DA	R-S	BOF	CONE	*	*	OUTW	*	LST	*	*	*	CJP	≠INT	-	ADDR(IF+0)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(INTR+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
SBB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(22)
+ 3	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(SBB4+3)
+ 4	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(SR+2)
SBB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	*	*	CJP	YES	-	ADDR(SBB4+2)
SBB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	*	*	CJP	YES	-	ADDR(SBB4+2)
SBB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(70)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	*	*	CJP	YES	-	ADDR(SBB4+2)
SI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(SBB4+3)
SIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(SR+2)
S+ 0	DO	ADD	OF	CONE	*	*	*	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(SBB4+3)
DSR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	OB+	OR	OF	*	*	*	*	INX	*	*	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	OIC	MRRQ	IRKJ	INB	*	*	*	
+ 3	DA+	R-S	BOF	CONE	*	*	OUTX	*	LDZO	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	DA	R-S	BOF	CLNK	*	*	OUTW	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DS+ 0	DO	ADD	OF	CONE	*	*	*	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(DS+2)
+ 3	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(DS+4)
+ 4	*	*	*	*	*	*	OMDR	INX	*	*	*	*	JRP	GO=0	-	ADDR(DSR+2)
DSI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	*	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(DS+2)
MSR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	DO	AND	QOF	*	*	*	*	*	OIC	MRRQ	IRKJ	INB	*	*	*	
+ 2	DO	OR	OF	*	*	*	OUTW	INX	*	*	*	*	CJP	WO=0	-	ADDR(MSR+4)
+ 3	DO	R-S	OF	CONE	*	*	OUTX	INX	ISGN	*	*	*	*	*	*	
+ 4	OA	OR	OF	*	*	*	*	INZ	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 5	DO	AND	OF	*	*	*	*	INY	*	*	*	*	CJP	FO=0	-	ADDR(MSR+7)
+ 6	OA	S-R	OF	CONE	*	*	*	INZ	ISGN	*	*	*	*	*	*	
+ 7	DO	AND	BOF	*	*	*	OJK	INN	*	*	PLJK	*	*	*	*	uDS(OF)
+ 8	*	*	*	*	SZRO	RW	*	*	*	*	*	*	CJP	≠X15	-	ADDR(MSR+11)
+ 9	DA	ADD	BOF	CZRO	*	*	OUTZ	*	*	*	*	*	*	*	*	
+10	DQ	ADD	QOF	*	*	*	OUTY	*	*	*	*	*	CJP	OVR	-	ADDR(MSR+15)
+11	OQ	OR	OF	*	SZRO	LY	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(MSR+8)
+12	OA	OR	OF	*	*	*	*	*	*	*	*	*	CJP	F≠0	-	ADDR(MSR+15)
+13	OA	OR	OF	*	*	*	*	*	LST	*	*	*	CJP	≠SGN	-	ADDR(MSR+16)
+14	OA	S-R	BOF	CONE	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+15	DO	AND	BOF	*	*	*	*	*	LSTV	*	*	*	*	*	*	
+16	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
MISP+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISP+0)
+ 1	DO	AND	QOF	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(MSR+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
MISN+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISN+0)
+ 1	DO	AND	QOF	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(MSR+2)
MSIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+0)
+ 1	DO	AND	QOF	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(MSR+2)
MB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	AND	OF	CZRO	*	*	OJK	INEA			IRJK	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	OTOA	MRRQ	PLJK	INAB	*	*	*	uDS(22)
+ 3	DO	OR	OF	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(MB4+3)
+ 4	DO	AND	QOF	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(MSR+2)
MB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	_	ADDR(MB4+2)
MB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(MB4+2)
MB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(70)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(MB4+2)
MS+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(MB4+3)
MSI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(MB4+3)
M+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 2	DO	OR	OF	*	*	*	OMDR	ININ	*	*	*	*	CJP	GO=0	-	ADDR(M+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
M+ 3	DO	AND	QOF	*	*	*	*	*	OIC	MRRQ	*	*	*	*	*	
+ 4	DO	OR	OF	*	*	*	OUTW	INX	*	*	*	*	CJP	WO=0	-	ADDR(M+6)
+ 5	DO	R-S	OF	CONE	*	*	OUTX	INX	ISGN	*	*	*	*	*	*	
+ 6	OA	OR	OF	*	*	*	*	INZ	*	*	*	*	LDCT	*	-	ADDR(M+10)
+ 7	OD	AND	OF	*	*	*	*	INY	*	*	*	*	CJP	FO=0	-	ADDR(M+9)
+ 8	OA	S-R	OF	CONE	*	*	*	INZ	*	*	*	*	*	*	*	
+ 9	OD	AND	BOF	*	*	*	*	INN	*	*	IRJK	*	*	*	*	uDS(OF)
+10	*	*	*	*	SZRO	RW	*	*	*	*	*	*	CJP	X15	-	ADDR(M+13)
+11	DA+	ADD	BOF	CZRO	*	*	OUTZ	*	*	*	*	*	*	*	*	
+12	DA	ADD	BOF	CLNK	SZRO	LY	OUTY	*	DECN	*	*	*	JRP	N≠0	-	ADDR(M+14)
+13	*	*	*	*	SZRO	LY	*	*	DECN	*	*	*	JRP	N≠0	-	ADDR(M+14)
+14	*	*	*	*	*	*	*	*	*	*	*	*	*	≠SGN	-	ADDR(M+17)
+15	OA+	S-R	BOF	CONE	*	*	*	*	LDZO	*	*	*	LDCT	*	-	ADDR(IF+0)
+16	OA	S-R	BOF	CLNK	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+17	OA+	OR	OF	*	*	*	*	*	LDZO	*	*	*	LDCT	*	-	ADDR(IF+0)
+18	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
MR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	DO	AND	QOF	*	*	*	*	*	OIC	MRRQ	IRKJ	INB	CJP	YES	-	ADDR(M+4)
MI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(M+2)
MIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	DO	AND	QOF	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(M+4)
DM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	*	*	*	
+ 2	DO	OR	OF	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(DM+2)
+ 3	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(DM+12)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DM+ 4	*	*	*	1	*	*	OMDR	INX	*	*	*	*	CJP	GO=0	-	ADDR(DM+4)
+ 5	OA	OR	OF	*	*	*	*	*	OIC	MRRQ	*	*	*	WO=0	-	ADDR(DM+9)
+ 6	DO	R-S	OF	CONE	*	*	OUTX	INX	ISGN	*	*	*	*	*	*	
+ 7	DO	R-S	OF	CLNK	*	*	OUTW	INW	*	*	*	*	*	*	*	
+ 8	OA	OR	OF	*	*	*	*	*	*	*	*	*	*	*	*	
+ 9	OA+	OR	QOF	*	*	*	*	*	*	*	IRJK	INB	CJP	FO=0	-	ADDR(DM+12)
+10	OA+	S-R	QOF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+11	OA	S-R	BOF	CLNK	*	*	*	*	*	*	*	*	*	*	*	
+12	*	*	*	*	*	*	OJK	INN	*	*	PLJK	*	*	*	*	uDS(20)
+13	OA	OR	OF	*	SZRO	RW	*	*	DECN	*	*	*	CJP	≠X15	-	ADDR(DM+18)
+14	DQ	ADD	OF	CZRO	*	*	OUTZ	INZ	*	*	*	*	CJP	FO=1	-	ADDR(DM+17)
+15	DA	ADD	OF	CLNK	*	*	OUTY	INY	*	*	*	*	*	*	*	
+16	*	*	*	*	*	*	*	*	*	*	*	*	CJP	≠OVR	-	ADDR(DM+19)
+17	*	*	*	*	*	*	*	*	LSTV	*	*	*	CJP	YES	-	ADDR(DM+22)
+18	DO	OR	OF	*	*	*	OUTW	*	*	*	*	*	CJP	FO=1	-	ADDR(DM+20)
+19	OA	OR	LBQ	*	SZRO	*	*	*	*	*	*	*	JRP	N≠0	-	ADDR(DM+24)
+20	DO	OR	OF	*	*	*	OUTX	*	*	*	*	*	CJP	F≠0	-	ADDR(DM+22)
+21	*	*	*	*	*	*	*	*	LSTV	*	*	*	CJP	F=0	-	ADDR(DM+25)
+22	*	*	*	*	*	*	*	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+23	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR)
+24	*	*	*	*	*	*	*	*	*	*	*	*	*	≠SGN	-	ADDR(DM+25)
+25	DO+	R-S	BOF	CONE	*	*	OUTZ	*	DZRO	*	*	*	LDCT	*	-	ADDR(IF+0)
+26	DO	R-S	BOF	CLNK	*	*	OUTY	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR)
+27	DO+	OR	BOF	*	*	*	OUTZ	*	DZRO	*	*	*	LDCT	*	-	ADDR(IF+0)
+28	DO	OR	BOF	*	*	*	OUTY	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR)
DMR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	*	INAB	LDCT	*	-	ADDR(DM+12)
+ 1	OB+	OR	OF	*	*	*	*	INX	*	*	*	*	CJP	YES	-	ADDR(DM+5)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DMI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(DM+2)
DVR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	IRKJ	INB	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	WO=0	-	ADDR(DVR+4)
+ 3	DO	R-S	OF	CONE	*	*	OUTW	INW	ISGN	*	*	*	*	*	*	
+ 4	OA	OR	QOF	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(DVR+26)
+ 5	OQ+	OR	BOF	*	*	*	*	*	*	*	*	*	CJP	FO=0	-	ADDR(DVR+8)
+ 6	OA	S-R	BOF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+ 7	OA	R-S	OF	CONE	*	*	OUTW	*	*	*	*	*	*	*	*	
+ 8	DO	AND	OF	*	*	*	*	ING	*	*	*	*	CJP	FO=1	-	ADDR(DVR+25)
+ 9	*	*	*	*	*	*	*	*	*	*	*	*	CJP	W1=1	-	ADDR(DVR+11)
+10	DO	ADD	OF	CONE	SZRO	LW	OUTG	ING	*	*	*	*	CJP	YES	-	ADDR(DVR+9)
+11	*	*	*	*	*	*	OUTG	INN	*	*	*	*	LDCT	*	-	ADDR(DVR+13)
+12	DA	R-S	BOF	CONE	*	*	OUTW	*	*	*	*	*	CJP	N=0	-	ADDR(DVR+16)
+13	*	*	*	*	SZRO	RW	*	*	DECN	*	*	*	CJP	FO=1	-	ADDR(DVR+15)
+14	DA	R-S	BOF	CONE	SONE	LY	OUTW	*	*	*	*	*	JRP	N≠0	-	ADDR(DVR+16)
+15	DA	ADD	BOF	CZRO	SZRO	LY	OUTW	*	*	*	*	*	JRP	N≠0	-	ADDR(DVR+16)
+16	*	*	*	*	*	*	*	*	*	*	*	*	CJP	FO=1	-	ADDR(DVR+18)
+17	OA	OR	OF	*	SONE	LY	*	INW	*	*	*	*	CJP	YES	-	ADDR(DVR+19)
+18	OA	ADD	BOF	CZRO	SZRO	LY	OUTW	INW	*	*	*	*	*	*	*	
+19	OA+	OR	OF	*	*	*	*	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+20	DO	OR	BOF	*	*	*	OUTZ	*	LST	*	*	*	CJP	≠SGN	-	ADDR(DVR+22)
+21	OA	R-S	BOF	CONE	*	*	*	*	LST	*	*	*	*	*	*	
+22	DO+	OR	BOF	*	*	*	OUTW	*	*	*	*	*	CJP	FO=0	-	ADDR(DVR+24)
+23	DO+	R-S	BOF	CONE	*	*	OUTW	*	*	*	*	*	*	*	*	
+24	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR)
+25	DO	AND	BOF	*	*	*	*	*	LST	*	*	*	CJP	YES	-	ADDR(DVR+24)
+26	*	*	*	*	*	*	*	*	LSTV	*	*	*	CJP	YES	-	ADDR(DVR+24)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DISP+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISP+0)
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(DVR+12)
DISN+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISN+0)
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(DVR+12)
DVIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(DVR+12)
DB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(22)
+ 3	DO	OR	OF	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=1	-	ADDR(DB4+3)
+ 4	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(DVR+12)
DB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(DB4+3)
DB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(DB4+3)
DB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(70)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(DB4+3)
DV+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(DB4+3)
DI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(DB4+3)

AD-A053 345 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
DESIGN OF THE PROCESSOR FOR SOFTWARE COMPATIBLE AVIONIC COMPUTE--ETC(U)
DEC 77 F G THOUROT
UNCLASSIFIED AFIT/GCS/EE/77-10 NL

3 OF 3
AD
A053345

END
DATE
FILMED
6-78
DDC

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	DO	OR	OF	*	*	*	OUTW	*	OIC	MRRQ	*	*	CJP	WO=0	-	ADDR(DR+3)
+ 2	DO	R-S	OF	CONE	*	*	OUTW	INW	ISGN	*	*	*	CJP	YES	-	ADDR(DR+4)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(DR+32)
+ 4	OA+	OR	QOF	*	*	*	*	*	*	*	IRJK	INB	*	*	*	
+ 5	OA	OR	OF	*	*	*	*	INY	*	*	*	*	*	*	*	
+ 6	*	*	*	*	*	*	*	*	*	*	*	*	CJP	FO=0	-	ADDR(DR+9)
+ 7	OQ	R-S	QOF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+ 8	OA	R-S	BOF	CLNK	*	*	*	*	*	*	*	*	*	*	*	
+ 9	DA	R-S	OF	CONE	*	*	*	*	*	*	*	*	LDCT	*	-	ADDR(DR+16)
+10	DO+	OR	BOF	*	*	*	OUTY	*	*	*	*	*	CJP	FO=0	-	ADDR(DR+32)
+11	DO	OR	OF	*	*	*	OJK	ING	CLYZ	*	PLJK	*	*	*	*	uDS(OF)
+12	DO	ADD	OF	CONE	SZRO	LW	OUTG	ING	*	*	*	*	CJP	W1=0	-	ADDR(DR+12)
+13	DO	S-R	OF	CZRO	SZRO	RW	OUTG	INN	*	*	*	*	CJP	YES	-	ADDR(DR+15)
+14	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
+15	DA	R-S	BOF	CZRO	*	*	OUTW	*	*	*	*	*	*	*	*	
+16	*	*	*	*	SZRO	RW	*	DECN	*	*	*	*	CJP	FO=1	-	ADDR(DR+19)
+17	DQ	R-S	QOF	CONE	SONE	LY	OUYX	*	*	*	*	*	*	*	*	
+18	DA	R-S	BOF	CLNK	*	*	OUTW	*	*	*	*	*	JRP	N≠0	-	ADDR(DR+21)
+19	DO	ADD	QOF	CZRO	SZRO	LY	OUTX	*	*	*	*	*	*	*	*	
+20	DA	ADD	BOF	CLNK	*	*	OUTW	*	*	*	*	*	JRP	N≠0	-	ADDR(DR+21)
+21	*	*	*	*	*	*	*	*	*	*	*	*	CJP	FO=0	-	ADDR(DR+25)
+22	DQ	ADD	QOF	CZRO	SZRO	LY	OUTX	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+23	DA	ADD	BOF	CLNK	*	*	OUTW	*	*	*	*	*	*	*	*	
+24	DA+	OR	OF	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(DR+26)
+25	DA+	OR	OF	*	SONE	LY	*	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+26	*	*	*	*	*	*	*	*	*	*	*	*	CJP	FO=1	-	ADDR(DR+28)
+27	OQ+	OR	BOF	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(DR+29)
+28	OQ+	R-S	BOF	CONE	*	*	*	*	*	*	*	*	*	*	*	
+29	DO	OR	BOF	*	*	*	OUTZ	*	LDS	*	*	*	CJP	≠SGN	-	ADDR(DR+31)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DR+30	DO	R-S	BOF	CONE	*	*	*	*	LDS	*	*	*	*	*	*	
+31	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR)
+32	*	*	*	*	*	*	*	*	LDSV	*	*	*	CJP	YES	-	ADDR(DR+31)
D+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	PUSH	*	*	
+ 2	DO	OR	OF	*	*	*	OMDR	INW	*	*	*	*	LOOP	GO=1	-	ADDR(DR+1)
DI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	PUSH	*	*	
+ 2	DO	OR	OF	*	*	*	OMDR	INW	*	*	*	*	LOOP	GO=1	-	ADDR(DR+1)
DIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	*	ADDR(DR+1)
DDR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	OB+	OR	OF	*	*	*	*	INX	*	*	*	*	*	*	*	
+ 2	OA	OR	OF	*	*	*	*	*	OIC	MRRQ	IRKJ	INB	*	*	*	
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F0=0	-	ADDR(DDR+6)
+ 4	OA+	S-R	QOF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+ 5	OA	S-R	BOF	CLNK	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(DDR+7)
+ 6	OA+	OR	QOF	*	*	*	*	*	*	*	*	*	*	*	*	
+ 7	*	*	*	*	*	*	*	*	*	*	*	*	CJP	W0=0	-	ADDR(DDR+10)
+ 8	DO	R-S	OF	CONE	*	*	OUTX	INX	ISGN	*	*	*	*	*	*	
+ 9	DO	R-S	OF	CLNK	*	*	OUTW	INW	*	*	*	*	CJP	YES	-	ADDR(DDR+13)
+10	DO	OR	OF	*	*	*	OUTX	*	LDZO	*	*	*	*	*	*	
+11	DO	OR	OF	*	*	*	OUTW	*	*	*	*	*	*	*	*	
+12	*	*	*	*	*	*	*	*	CLDZ	*	*	*	CJP	F=0	-	ADDR(DDR+34)
+13	DQ	R-S	OF	CONE	*	*	OUTX	*	*	*	*	*	*	*	*	
+14	DA	R-S	OF	CLNK	*	*	OUTW	*	*	*	*	*	*	*	*	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DDR+15	DO	AND	OF	*	*	*	*	ING	*	*	*	*	CJP	FO=1	-	ADDR(DDR+35)
+16	DO	ADD	OF	CONE	SZRO	LW	OUTG	ING	*	*	*	*	CJP	W1=0	-	ADDR(DDR+16)
+17	DO	S-R	OF	CZRO	SZRO	RW	OUTG	INN	*	*	*	*	*	*	*	
+18	DQ	R-S	QOF	CONE	*	*	OUTX	*	*	*	*	*	LDCT	*	-	ADDR(DDR+20)
+19	DA	R-S	BOF	CLNK	*	*	OUTW	*	*	*	*	*	CJP	N=0	-	ADDR(DDR+25)
+20	*	*	*	*	SZRO	RW	*	*	DECN	*	*	*	CJP	FO=1	-	ADDR(DDR+23)
+21	DQ	S-R	QOF	CONE	SONE	LY	OUTX	*	*	*	*	*	*	*	*	
+22	DA	S-R	BOF	CLNK	*	*	OUTW	*	*	*	*	*	JPR	N≠0	-	ADDR(DDR+25)
+23	DQ	ADD	QOF	CZRO	SZRO	LY	OUTX	*	*	*	*	*	*	*	*	
+24	DA	ADD	BOF	CLNK	*	*	OUTW	*	*	*	*	*	JRP	N≠0	-	ADDR(DDR+25)
+25	*	*	*	*	*	*	*	*	*	*	*	*	CJP	FO=1	-	ADDR(DDR+27)
+26	*	*	*	*	SONE	LY	*	*	*	*	*	*	CJP	YES	-	ADDR(DDR+28)
+27	*	*	*	*	SZRO	LY	*	*	*	*	*	*	*	*	*	
+28	*	*	*	*	*	*	*	*	*	*	*	*	CJP	SGN	-	ADDR(DDR+31)
+29	DO+	OR	BOF	*	*	*	OUTZ	*	LDZO	*	*	*	LDCT	*	-	ADDR(IF+0)
+30	DO	OR	BOF	*	*	*	OUTY	*	LST	*	*	*	CJP	YES	-	ADDR(DDR+33)
+31	DO+	S-R	BOF	CONE	*	*	OUTZ	*	LDZO	*	*	*	LDCT	*	-	ADDR(IF+0)
+32	DO	S-R	BOF	CLNK	*	*	OUTY	*	LST	*	*	*	*	*	*	
+33	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+34	*	*	*	*	*	*	*	*	LSTV	*	*	*	CJP	YES	-	ADDR(DDR+33)
+35	DO	AND	BOF	*	*	*	*	*	*	*	*	*	*	*	*	
+36	DO+	AND	BOF	*	*	*	*	*	LST	*	*	*	CJP	YES	-	ADDR(DDR+33)
DD+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	PUSH	*	*	
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	LOOP	GO=1	*	
+ 3	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(DD+4)
+ 4	*	*	*	*	*	*	OMDR	INX	*	*	*	*	JRP	GO=0	-	ADDR(DDR+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DDI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(DDI+2)
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	JRP	GO=1	-	ADDR(DD+3)
IM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	OR	QOF	*	*	*	OJK	*	OEAR	MRRQ	PLJK	*	*	*	*	uDS(OF)
+ 2	DO	AND	QOF	*	*	*	OJK	*	*	*	IRJK	*	*	*	*	
+ 3	DO	OR	OF	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(IM+3)
+ 4	DO	ADD	QOF	CONE	*	*	OUTW	*	LST	*	*	*	*	*	*	
+ 5	OQ	OR	OF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 6	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(IM+6)
+ 7	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
DMM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	OR	QOF	*	*	*	OJK	*	OEAR	MRRQ	PLJK	*	*	*	*	uDS(OF)
+ 2	DQ	AND	QOF	*	*	*	OJK	*	*	*	IRKJ	*	*	*	*	
+ 3	DO	OR	OF	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	*	ADDR(DMM+3)
+ 4	DQ	S-R	QOF	CZRO	*	*	OUTW	*	LST	*	*	*	CJP	YES	*	ADDR(IM+5)
ABS+ 0	OA	OR	OF	*	*	*	*	*	OIC	MRRQ	IRKJ	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	FO=1	-	ADDR(NEG+1)
+ 2	OA	OR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
NEG+ 0	*	*	*	*	*	*	*	*	OIC	MRRQ	IRKJ	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	OA	R-S	BOF	CONE	*	*	*	*	LST	*	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DABS+ 0	OA	OR	OF	*	*	*	*	*	OIC	MRRQ	IRKJ	INAB	*	*	*	
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	FO=1	-	ADDR(DNEG+1)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DABS+ 2	OA+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	OA	OR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DNEG+ 0	*	*	*	*	*	*	*	*	OIC	MRRQ	IRKJ	INAB	*	*	*	
+ 1	OA+	S-R	BOF	CONE	*	*	*	*	LDZO	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	OA	S-R	BOF	CLNK	*	*	*	*	LST	*	*	*	*	*	*	
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
CR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	DA	R-S	OF	CONE	*	*	OUTW	*	LST	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
C+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(CR+1)
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	JRP	GO=1	-	ADDR(C+2)
CI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(CR+1)
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	JRP	GO=1	-	ADDR(CI+2)
CISP+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISP+0)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(CR+1)
CISN+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISN+0)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(CR+1)
CIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(CR+1)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DCR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	*	*	*	*	
+ 1	OB+	OR	OF	*	*	*	*	INX	OIC	MRRQ	*	*	*	*	*	
+ 2	DA+	R-S	OF	CONE	*	*	OUTX	*	DZRO	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	DA	R-S	OF	CLNK	*	*	OUTX	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DC+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(DC+2)
+ 3	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 4	*	*	*	*	*	*	OMDR	INX	*	*	*	*	CJP	GO=0	-	ADDR(DC+4)
+ 5	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(DCR+2)
DCI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(DC+2)
FCR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	OB+	OR	OF	*	*	*	*	INX	*	*	*	*	*	*	*	
+ 2	DA	XOR	OF	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	OA	OR	OF	*	*	*	*	*	LST	*	*	*	CJP	F0=1	-	ADDR(FCR+16)
+ 4	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(00FF)
+ 5	DQ	AND	OF	*	*	*	OUTX	ING	*	*	*	*	*	*	*	
+ 6	AQ	AND	QOF	*	*	*	*	*	*	*	*	*	*	*	*	
+ 7	DQ	R-S	QOF	*	*	*	*	*	*	*	*	*	*	*	*	
+ 8	OQ	OR	OF	*	*	*	*	*	*	*	*	*	CJP	F0=1	-	ADDR(FCR+15)
+ 9	OA	OR	OF	*	*	*	*	*	LST	*	*	*	CJP	F≠0	-	ADDR(FCR+16)
+10	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(FF00)
+11	DQ	AND	OF	*	*	*	OUTX	INX	*	*	*	*	*	*	*	
+12	AQ+	AND	QOF	*	*	*	*	*	*	*	*	*	*	*	*	
+13	DQ	R-S	OF	CONE	*	*	OUTX	*	DZRO	*	*	*	*	*	*	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
FCR+14	DA	R-S	OF	CLNK	*	*	OUTW	*	LST	*	*	*	CJP	YES	-	ADDR(FCR+16)
+15	OA	S-R	OF	*	*	*	*	*	LST	*	*	*	*	*	*	
+16	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
FC+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(FC+2)
+ 3	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(FCR+2)
+ 4	*	*	*	*	*	*	OMDR	INX	*	*	*	*	JRP	GO=1	-	ADDR(FC+4)
FCI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(FC+2)
SCAL+ 0	DO	OR	OF	*	*	*	OUTX	ING	*	*	*	*	*	*	*	
+ 1	DO	OR	QOF	*	*	*	OUTG	INZ	*	*	*	*	*	*	*	
+ 2	OA+	OR	OF	*	*	*	*	ING	*	*	*	*	*	*	*	
+ 3	DQ	S-R	QOF	CONE	*	*	OUTG	*	*	*	*	*	*	*	*	
+ 4	OQ	OR	OF	*	*	*	*	*	*	*	*	*	CJP	FO=1	-	ADDR(SCAL+13)
+ 5	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(SCAL+21)
+ 6	DQ	R-S	OF	CONE	*	*	OJK	*	*	*	PLJK	*	*	*	*	uDS(18)
+ 7	OQ	R-S	OF	CZRO	*	*	*	INN	*	*	*	*	CJP	FO=0	--	ADDR(SCAL+24)
+ 8	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(FF00)
+ 9	DQ	AND	OF	*	*	*	OUTX	INX	*	*	*	*	*	*	*	
+10	AQ+	AND	QOF	*	*	*	*	*	*	*	*	*	*	*	*	
+11	*	*	*	*	SWO	RW	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SCAL+11)
+12	DO+	OR	BOF	*	*	*	OUTG	*	*	*	*	*	CRTN	YES	*	
+13	OQ	S-R	QOF	CONE	*	*	*	*	*	*	*	*	*	*	*	
+14	DQ	R-S	OF	CONE	*	*	OJK	*	*	*	PLJK	*	*	-	-	uDS(18)
+15	OQ	R-S	OF	CZRO	*	*	*	INN	*	*	*	*	CJP	FO=0	-	ADDR(SCAL+27)
+16	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(FF00)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
SCAL+17	DQ	AND	OF	*	*	*	OUTX	INX	*	*	*	*	*	*	*	
+18	AQ+	AND	QOF	*	*	*	*	*	*	*	*	*	*	*	*	
+19	OA	OR	RBQ	*	SOBO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SCAL+19)
+20	DO+	OR	BOF	*	*	*	OUTZ	*	*	*	*	*	CRTN	YES	*	
+21	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(FF00)
+22	DQ	AND	OF	*	*	*	OUTX	INX	*	*	*	*	*	*	*	
+23	AQ+	AND	QOF	*	*	*	*	*	*	*	*	*	CJP	YES	*	ADDR(SCAL+25)
+24	OA+	AND	QOF	*	*	*	*	*	CLWX	*	*	*	*	*	*	uDL(FF00)
+25	DO+	OR	BOF	*	*	*	OUTG	*	*	*	*	*	CRTN	YES	*	
+26	DO	AND	BOF	*	*	*	*	*	*	*	*	*	*	*	*	
+27	DO	OR	QOF	*	*	*	*	*	*	*	*	*	*	*	-	uDL(FF00)
+28	DQ	AND	OF	*	*	*	OUTX	INX	*	*	*	*	*	*	*	
+29	DO+	OR	BOF	*	*	*	OUTZ	*	*	*	*	*	*	*	*	
+30	DO	AND	QOF	*	*	*	*	*	*	*	*	*	CRTN	YES	*	
NORM+ 0	DO	OR	QOF	*	*	*	OUTW	*	*	*	*	*	CJP	NRM	-	ADDR(NORM+5)
+ 1	DQ	OR	OF	*	*	*	OUTX	*	*	*	*	*	*	*	*	
+ 2	OA+	R-S	BOF	CZRO	SZRO	LW	*	*	*	*	*	*	CJP	F=0	-	ADDR(NORM+20)
+ 3	OA+	OR	OF	*	*	*	*	*	*	*	*	*	CJP	NRM	-	ADDR(NORM+5)
+ 4	OA+	R-S	BOF	CZRO	SZRO	LW	*	*	*	*	*	*	CJP	YES	-	ADDR(NORM+3)
+ 5	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F0=1	-	ADDR(NORM+11)
+ 6	DA+	R-S	OF	CONE	*	*	uDL	*	*	*	*	*	*	-	-	uDL(007F)
+ 7	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F0=1	-	ADDR(NORM+13)
+ 8	DO	AND	QOF	*	*	*	*	*	LSTV	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 9	DO+	OR	BOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(0080)
+10	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+11	DA+	ADD	OF	CZRO	*	*	uDL	*	*	*	*	*	*	-	-	uDL(0080)
+12	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F0=1	-	ADDR(NORM+18)
+13	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(FF00)
+14	DQ	AND	OF	*	*	*	OUTX	INX	*	*	*	*	*	*	*	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
NORM+15	DA+	AND	BOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(00FF)
+16	DO	OR	BOF	*	*	*	OUTW	*	LST	*	*	*	LDCT	*	-	ADDR(IF+0)
+17	DA+	OR	BOF	*	*	*	OUTX	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+18	DO	AND	BOF	*	*	*	*	*	LSTU	*	*	*	LDCT	*	-	ADDR(IF+0)
+19	DO+	OR	BOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(0080)
+20	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+21	DO	AND	BOF	*	*	*	*	*	LST	*	*	*	LDCT	*	-	ADDR(IF+0)
+22	DO+	OR	BOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(0080)
+23	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
FAR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	OB+	OR	OF	*	*	*	*	INX	*	*	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	IRKJ	INB	CJP	YES	-	ADDR(FAB4+6)
FA+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(FAB4+3)
FAI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(FAB4+3)
FAB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	*	*	*	
+ 2	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	PLJK	INAB	*	*	*	uDS(00)
+ 3	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=1	-	ADDR(FAB4+3)
+ 4	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 5	*	*	*	*	*	*	OMDR	INX	*	*	*	*	CJP	GO=0	-	ADDR(FAB4+5)
+ 6	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJS	YES	-	ADDR(SCAL+0)
+ 7	AD+	ADD	OF	CONE	*	*	OUTX	INX	*	*	*	*	*	*	*	
+ 8	AD	ADD	OF	CLNK	*	*	OUTW	INW	*	*	*	*	*	*	*	
+ 9	*	*	*	*	*	*	*	*	*	*	*	*	CJP	≠OVR	-	ADDR(NORM+0)
+10	OA+	ADD	BOF	CONE	SLNK	RW	*	*	*	*	*	*	CJP	YES	-	ADDR(NORM+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
FAB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FAB4+2)
FAB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FAB4+2)
FAB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(70)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FAB4+2)
FSR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	OB+	OR	OF	*	*	*	*	INX	*	*	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	IRKJ	INB	CJP	YES	-	ADDR(FSB4+6)
FS+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(FSB4+3)
FSI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(FSB4+3)
FSB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	*	*	*	
+ 2	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	PLJK	INAB	*	*	*	uDS(00)
+ 3	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(FSB4+3)
+ 4	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 5	*	*	*	*	*	*	OMDR	INX	*	*	*	*	CJP	GO=0	-	ADDR(FSB4+5)
+ 6	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJS	YES	-	ADDR(SCAL+0)
+ 7	DA+	R-S	OF	CONE	*	*	OUTX	INX	*	*	*	*	*	*	*	
+ 8	DA	R-S	OF	CLNK	*	*	OUTW	INW	*	*	*	*	*	*	*	
+ 9	*	*	*	*	*	*	*	*	*	*	*	*	CJP	≠OVR	-	ADDR(NORM+0)
+10	OA+	ADD	BOF	CONE	SLNK	RW	*	*	*	*	*	*	CJP	YES	*	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
FSB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FSB4+2)
FSB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FSB4+2)
FSB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(70)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FSB4+2)
FMR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	OB+	OR	OF	*	*	*	*	INX	*	*	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	IRKJ	INB	CJP	YES	-	ADDR(FMB4+6)
FM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(FMB4+3)
FMI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(FMB4+3)
FMB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	*	*	*	
+ 2	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	PLJK	INAB	*	*	*	uDS(00)
+ 3	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(FMB4+3)
+ 4	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 5	*	*	*	*	*	*	OMDR	INX	*	*	*	*	CJP	GO=0	-	ADDR(FMB4+5
+ 6	DO	OR	QOF	*	*	*	uDL	*	OIC	MRRQ	*	*	*	-	-	uDL(FF00)
+ 7	DO	OR	OF	*	*	*	OUTX	ING	*	*	*	*	*	*	*	
+ 8	DQ	AND	OF	*	*	*	OUTX	INX	*	*	*	*	*	*	*	
+ 9	AQ+	AND	OF	*	*	*	*	INZ	*	*	*	*	*	*	*	
+10	DO+	OR	BOA	*	*	*	OUTG	ING	*	*	*	*	*	*	*	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
FMB4+11	DA+	ADD	BOF	CONE	*	*	OUTG	*	*	*	*	*	*	*	*	
+12	OA	OR	OF	*	*	*	*	INY	*	*	*	*	*	*	*	
+13	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F0=1	-	ADDR(FMB4+25)
+14	DO	OR	OF	*	*	*	OUTW	*	*	*	*	*	CJP	F=0	-	ADDR(FMB4+20)
+15	DO	OR	OF	*	*	*	OUTW	*	*	*	*	*	CJP	WO=1	-	ADDR(FMB4+28)
+16	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(FMB4+30)
+17	DO	AND	BOF	*	SZRO	RY	*	*	*	*	*	*	LDCT	*	-	ADDR(FMB4+19)
+18	DO	AND	QOF	*	SZRO	RY	*	INN	*	*	*	*	*	*	*	uDS(18)
+19	DQ	ADD	QOF	CZRO	SZRO	LW	OUTZ	*	DECN	*	*	*	CJP	WO=1	-	ADDR(FMB4+21)
+20	DQ	R-S	QOF	CONE	SZRO	RY	OUTZ	*	*	*	*	*	JRP	N≠0	-	ADDR(FMB4+22)
+21	DA	ADD	BOF	CLNK	SZRO	RY	OUTY	*	*	*	*	*	JRP	N=0	-	ADDR(FMB4+22)
+22	OA+	ADD	BOF	CONE	*	*	*	*	*	*	*	*	CJP	SGN	-	ADDR(FMB4+33)
+23	OQ	OR	OF	*	*	*	*	INX	*	*	*	*	*	*	*	
+24	OA	OR	OF	*	*	*	*	INW	*	*	*	*	CJP	YES	-	ADDR(NORM+0)
+25	DO	R-S	OF	CONE	*	*	OUTZ	INZ	ISGN	*	*	*	*	*	*	
+26	DO	R-S	OF	CLNK	*	*	OUTY	INY	*	*	*	*	*	*	*	
+27	DO	OR	OF	*	*	*	OUTW	*	*	*	*	*	CJP	YES	-	ADDR(FMB4+15)
+28	DO	R-S	OF	CONE	*	*	OUTX	INX	ISGN	*	*	*	*	*	*	
+29	DO	R-S	OF	CLNK	*	*	OUTW	INW	*	*	*	*	CJP	YES	-	ADDR(FMB4+17)
+30	DO	AND	BOF	*	*	*	*	*	LST	*	*	*	LDCT	*	-	ADDR(IF+0)
+31	DO+	OR	BOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(0080)
+32	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
FMB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	OA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FMB4+2)
FMB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	
+ 1	OA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FMB4+2)
FMB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	
+ 1	OA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FMB4+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
FDR+ 0	OB	OR	OF	*	*	*	*	INW	*	*	IRJK	INAB	*	*	*	
+ 1	OB+	OR	OF	*	*	*	*	INX	*	*	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	IRKJ	INB	CJP	YES	-	ADDR(FDB4+6)
FD+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(FDB4+3)
FDI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(FDB4+3)
FDB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	*	*	*	uDS(00)
+ 2	DO	ADD	OF	CONE	*	*	ATOD	INEA	OFAR	MRRQ	PLJK	INAB	*	*	*	
+ 3	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(FDB4+3)
+ 4	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 5	*	*	*	*	*	*	OMDR	INX	*	*	*	*	CJP	GO=0	-	ADDR(FDB4+5)
+ 6	DO	OR	QOF	*	*	*	uDL	*	OIC	MRRQ	*	*	*	-	-	uDL(FF00)
+ 7	DO	CR	OF	*	*	*	OUTX	ING	*	*	*	*	*	*	*	
+ 8	DQ	AND	OF	*	*	*	OUTX	INX	*	*	*	*	*	*	*	
+ 9	AQ+	AND	OF	*	*	*	*	INZ	*	*	*	*	*	*	*	
+10	DO+	OR	BOA	*	*	*	OUTG	ING	*	*	*	*	*	*	*	
+11	DA+	S-R	BOF	CONE	*	*	OUTG	*	*	*	*	*	*	*	*	
+12	DO	OR	QOF	*	*	*	OUTZ	*	*	*	*	*	*	*	*	
+13	OA	OR	RBQ	*	SOBO	*	*	*	*	*	*	*	*	*	*	
+14	OA+	ADD	BOF	CONE	*	*	*	*	*	*	*	*	*	*	*	
+15	DO	OR	OF	*	*	*	OUTW	*	*	*	*	*	CJP	WO=1	-	ADDR(FDB4+20)
+16	AO	OR	OF	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(FDB4+25)
+17	AO	OR	OF	*	*	*	*	*	*	*	*	*	CJP	FO=1	-	ADDR(FDB4+23)
+18	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(FDB4+26)
+19	OA	OR	RBQ	*	SZRO	RW	*	*	*	*	*	*	*	*	*	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
FDB4+20	DO	R-S	OF	CONE	*	*	OUTX	INX	ISGN	*	*	*	*	*	*	
+21	DO	R-S	OF	CLNK	*	*	OUTW	INW	*	*	*	*	*	*	*	
+22	OA	OR	OF	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(FDB4+17)
+23	OQ	*	QOF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+24	OA	S-R	BOF	CLNK	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(FDB4+19)
+25	DO	AND	BOF	*	*	*	*	*	LSTV	*	*	*	CJP	YES	-	ADDR(FDB4+27)
+26	DO	AND	BOF	*	*	*	*	*	LST	*	*	*	*	*	*	
+27	DO+	OR	BOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(0080)
+28	*	*	*	*	*	*	*	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+29	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+30	*	*	*	*	*	*	uDL	INN	*	*	*	*	*	-	-	uDL(0020)
+31	DQ	R-S	QOF	CONE	*	*	OUTX	*	*	*	*	*	*	*	*	
+32	DA	R-S	LBQ	CLNK	SZRO	*	*	*		*	*	*	LDCT	*	-	ADDR(FDB4+33)
+33	*	*	*	*	*	*	*	*	DECN	*	*	*	CJP	F0=1	-	ADDR(FDB4+36)
+34	DQ	R-S	QOF	CONE	SONE	LY	OUTX	*	*	*	*	*	*	*	*	
+35	DA	R-S	LBQ	CLNK	SZRO	*	OUTW	*	*	*	*	*	JRP	N≠0	-	ADDR(FDB4+38)
+36	DQ	ADD	QOF	CONE	SZRO	LY	OUTX	*	*	*	*	*	*	*	*	
+37	DA	ADD	LBQ	CLNK	SZRO	*	OUTW	*	*	*	*	*	JRP	N≠0	-	ADDR(FDB4+38)
+38	DO	OR	OF	*	*	*	OUTY	*	*	*	*	*	*	*	*	
+39	*	*	*	*	*	*	*	*	*	*	*	*	CJP	SGN	-	ADDR(FDB4+42)
+40	DO	OR	OF	*	*	*	OUTZ	INX	*	*	*	*	*	*	*	
+41	DO	OR	OF	*	*	*	OUTY	INW	*	*	*	*	CJP	YES	-	ADDR(NORM+0)
+42	DO	R-S	OF	CONE	*	*	OUTZ	INX	*	*	*	*	*	*	*	
+43	DO	R-S	OF	CLNK	*	*	OUTY	INW	*	*	*	*	CJP	YES	-	ADDR(NORM+0)
FDB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FDB4+2)
FDB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FDB4+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	1 BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
FDB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	-	uDS(70)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(FDB4+12)
FABS+ 0	OA	OR	OF	*	*	*	*	*	*	*	IRJK	INAB	*	*	*	
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F0=1	-	ADDR(FNEG+0)
+ 2	OA+	OR	BOF	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	OA	OR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
FNEG+ 0	DA+	AND	QOF	*	*	*	uDL	*	*	*	IRKJ	INAB	*	-	-	uDL(FF00)
+ 1	DA+	AND	BOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(00FF)
+ 2	OA	OR	RBQ	*	SOBO	*	*	*	*	*	*	*	*	*	*	
+ 3	OQ	S-R	OF	CONE	*	*	*	INX	*	*	*	*	*	*	*	
+ 4	OB	S-R	OF	CLNK	*	*	*	INW	OIC	MRRQ	*	*	*	*	*	
+ 5	OA	ADD	BOF	CONE	*	*	*	*	*	*	IRJK	INA	CJP	*	-	ADDR(NORM+0)
FLT+ 0	OB	OR	OF	*	*	*	*	INW	OIC	MRRQ	IRKJ	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(FLT+5)
+ 2	DO	OR	QOF	*	*	*	OJK	*	*	*	PLJK	*	*	*	*	uDS(OF)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	NRM	-	ADDR(FLT+7)
+ 4	OQ	R-S	QOF	CZRO	SZRO	LW	*	*	*	*	*	*	*	*	*	
+ 5	DO+	OR	BOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(0080)
+ 6	DO	AND	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+ 7	DQ+	AND	BOF	*	*	*	*	*	*	*	*	*	*	*	*	
+ 8	DO	OR	BOF	*	*	*	OUTW	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
FIX+ 0	OA	OR	OF	*	*	*	*	*	OIC	MRRQ	IRKJ	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	OA+	OR	OF	*	*	*	*	ING	*	*	*	*	CJP	F=0	-	ADDR(FIX+7)
+ 2	DO	OR	QOF	*	*	*	OJK	*	*	*	PLJK	*	*	*	*	uDS(OF)
+ 3	DQ	S-R	OF	CONE	*	*	OUTG	INW	*	*	*	*	CJP	F0=1	-	ADDR(FIX+8)
+ 4	DO	AND	QOF	*	*	*	*	*	*	*	*	*	CJP	F0=0	-	ADDR(FIX+9)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
FIX+ 5	DO	ADD	OF	CONE	*	*	OUTW	INW	*	*	*	*	CJP	WO=0	-	ADDR(FIX+10)
+ 6	OB	OR	RBQ	*	SOBO	*	*	*	*	*	*	*	CJP	YES	-	ADDR(FIX+5)
+ 7	DO	AND	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+ 8	DO	AND	BOF	*	*	*	*	*	LSTU	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+ 9	DO	AND	BOF	*	*	*	*	*	LSTV	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+10	OA	OR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
ANDR+ 0	*	*	*	*	*	*	*	*	OIC	MRRQ	IRKJ	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	AB	AND	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
AND+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(AND+2)
+ 3	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	DA	AND	BOF	*	*	*	OUTW	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
ANDI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(AND+2)
ANDM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(AND+3)
ORR+ 0	*	*	*	*	*	*	*	*	OIC	MRRQ	IRJK	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	AB	OR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
OR+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(OR+2)
+ 3	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	DA	OR	BOF	*	*	*	OUTW	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
ORI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(OR+2)
ORIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(OR+3)
XORR+ 0	*	*	*	*	*	*	*	*	OIC	MRRQ	IRJK	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	AB	XOR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
XOR+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(XOR+2)
+ 3	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	DA	XOR	BOF	*	*	*	OUTW	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
XORI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(XOR+2)
XORM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(XOR+3)
NR+ 0	AB	AND	BOF	*	*	*	*	*	OIC	MRRQ	IRKJ	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	OB	XNOR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
N+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(N+2)
+ 3	DA	AND	BOF	*	*	*	OUTW	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	OA	XNOR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
NI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(N+2)
NIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(N+3)
SLL+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRJK	INB	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	OJK	INW	OIC	MRRQ	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	DO	AND	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	
+ 3	OA	OR	LBQ	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SLL+3)
+ 4	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
SRL+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRJK	INB	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	OJK	*	OIC	MRRQ	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	DO	AND	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	
+ 3	OA	OR	RB	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SRL+3)
+ 4	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
SRA+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRJK	INB	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	OJK	*	OIC	MRRQ	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	DO	AND	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	
+ 3	OA	OR	RB	*	SOBO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SRA+3)
+ 4	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
SLC+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRJK	INB	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	OJK	INW	OIC	MRRQ	IRKJ	INA	LDCT	-	-	ADDR(IF+0)
+ 2	OB	OR	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	
+ 3	OA	OR	LBQ	*	SOBO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SLC+3)
+ 4	OQ	OR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
SRC+ 0	D0	OR	QOF	*	*	*	uDL	*	*	*	IRJK	INB	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	OJK	INW	OIC	MRRQ	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	OB	OR	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	
+ 3	OA	OR	RBQ	*	SQO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SRC+3)
+ 4	OQ	OR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DSLL+ 0	D0	OR	QOF	*	*	*	uDL	*	OIC	MRRQ	IRJK	INB	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	*	INW	*	*	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	OB+	OR	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	
+ 3	OB	OR	LBQ	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSLL+3)
+ 4	OQ+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+ 5	OB	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DSRL+ 0	D0	OR	QOF	*	*	*	uDL	*	OIC	MRRQ	IRJK	INB	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	*	INW	*	*	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	OB+	OR	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	
+ 3	OB	OR	RBQ	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSRL+3)
+ 4	OB+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+ 5	OB	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DSRC+ 0	D0	OR	QOF	*	*	*	uDL	*	OIC	MRRQ	IRJK	INB	*	-	-	uDL(00FF)
+ 1	DQ	AND	OF	*	*	*	*	INW	*	*	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	OB+	OR	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	
+ 3	OB	OR	RBQ	*	SQO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSRC+0)
+ 4	OQ+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+ 5	OB	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DSLC+ 0	D0	OR	QOF	*	*	*	uDL	*	OIC	MRRQ	IRJK	INB	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	*	INW	*	*	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	OB+	OR	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DSLC+ 3	OB	OR	LBQ	*	SRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSLC+3)
+ 4	OB+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+ 5	OB	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DSRA+ 0	DO	OR	QOF	*	*	*	uDL	*	OIC	MRRQ	IRJK	INB	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	*	INW	*	*	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	OB+	OR	QOF	*	*	*	OUTW	INN	*	*	*	*	*	*	*	
+ 3	OB	OR	RBQ	*	SOBO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSRA+3)
+ 4	OQ+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+ 5	OB	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
SLR+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRKJ	INA	*	-	-	uDL(0010)
+ 1	OB	OR	OF	*	*	*	*	*	OIC	MRRQ	IRJK	INB	LDCT	*	-	ADDR(IF+0)
+ 2	AQ	S-R	OF	CONE	*	*	*	INW	*	*	*	*	CJP	FO=0	-	ADDR(SLR+4)
+ 3	OB	S-R	BOF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+ 4	OB	OR	OF	*	*	*	OTOD	INN	*	*	IRJK	INA	CJP	WO=1	-	ADDR(SLR+11)
+ 5	DO	AND	BOF	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(SLR+8)
+ 6	DO	AND	QOF	*	*	*	*	*	*	*	IRKJ	INB	CJP	SGN	-	ADDR(SLR+9)
+ 7	OA	OR	LAQ	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SLR+7)
+ 8	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+ 9	OA	OR	RAQ	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SLR+9)
+10	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+11	OA	OR	OF	*	*	*	*	*	LSTC	*	*	*	*	*	*	
+12	DO	AND	BOF	*	*	*	*	*	*	*	IRJK	INB	JRP	≠INT	-	ADDR(INTR+0)
SAR+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRKJ	INA	*	-	-	uDL(0010)
+ 1	OB	OR	OF	*	*	*	*	*	OIC	MRRQ	IRJK	INB	LDCT	*	-	ADDR(IF+0)
+ 2	AQ	S-R	OF	CONE	*	*	*	INW	*	*	*	*	CJP	FO=0	-	ADDR(SAR+4)
+ 3	OB	S-R	BOF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+ 4	OB	OR	OF	*	*	*	OTOD	INN	*	*	IRJK	INA	CJP	WO=1	-	ADDR(SAR+11)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
SAR+ 5	DO	AND	BOF	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(SAR+8)
+ 6	DO	AND	QOF	*	*	*	*	*	*	*	IRKJ	INB	CJP	SGN	-	ADDR(SAR+9)
+ 7	OA	OR	LAQ	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SAR+7)
+ 8	OA	OR	OF	*	*	*	*	*	LST	*	*	*	CJP	≠INT	-	ADDR(INTR+0)
+ 9	OA	OR	RAQ	*	SOBO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SAR+9)
+10	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+11	OA	OR	OF	*	*	*	*	*	LSTC	*	*	*	*	*	*	
+12	DO	AND	BOF	*	*	*	*	*	*	*	IRJK	INB	JRP	≠INT	-	ADDR(INTR+0)
SCR+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRKJ	INA	*	-	-	uDL(0010)
+ 1	OB	OR	OF	*	*	*	*	*	OIC	MRRQ	IRJK	INB	LDCT	*	-	ADDR(IF+0)
+ 2	AQ	S-R	OF	CONE	*	*	*	INW	*	*	*	*	CJP	FO=0	-	ADDR(SCR+4)
+ 3	OB	S-R	BOF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+ 4	OB	OR	OF	*	*	*	OTOD	INN	*	*	IRJK	INA	CJP	WO=1	-	ADDR(SAR+11)
+ 5	DO	AND	BOF	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(SCR+8)
+ 6	DO	AND	QOF	*	*	*	*	*	*	*	IRKJ	INB	CJP	SGN	-	ADDR(SCR+9)
+ 7	OA	OR	LAQ	*	SRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SCR+7)
+ 8	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+ 9	OA	OR	RAQ	*	SQO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(SCR+9)
+10	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+11	OA	OR	OF	*	*	*	*	*	LSTC	*	*	*	*	*	*	
+12	DO	AND	BOF	*	*	*	*	*	*	*	IRJK	INB	JRP	≠INT	-	ADDR(INTR+0)
DSLR+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRKJ	INA	*	-	-	uDL(0020)
+ 1	OB	OR	OF	*	*	*	*	*	OIC	MRRQ	IRJK	INB	LDCT	*	-	ADDR(IF+0)
+ 2	AQ	S-R	OF	CONE	*	*	*	INW	*	*	*	*	CJP	FO=0	-	ADDR(DSLR+4)
+ 3	OB	S-R	OF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+ 4	OB	OR	OF	*	*	*	OTOD	INN	*	*	IRJK	INA	CJP	WO=1	-	ADDR(DSLR+12)
+ 5	DO	AND	BOF	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(DSLR+11)
+ 6	OB+	OR	QOF	*	*	*	*	*	*	*	IRKJ	INB	CJP	SGN	-	ADDR(DSLR+9)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DSLR+ 7	OA	OR	LAQ	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSLR+7)
+ 8	OA+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	CJP	YES	-	ADDR(DSLR+11)
+ 9	OA	OR	RAQ	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSLR+9)
+10	OQ+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+11	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+12	OA+	OR	OF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+13	OA	OR	OF	*	*	*	*	*	LST	*	*	*	*	*	*	
+14	DO	AND	BOF	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DSAR+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRKJ	INA	*	-	-	uDL(0020)
+ 1	OB	OR	OF	*	*	*	*	*	OIC	MRRQ	IRJK	INB	LDCT	*	-	ADDR(IF+0)
+ 2	AQ	S-R	OF	CONE	*	*	*	INW	*	*	*	*	CJP	F0=0	-	ADDR(DSAR+4)
+ 3	OB	S-R	OF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+ 4	OB	OR	OF	*	*	*	OTOD	INN	*	*	IRJK	INA	CJP	W0=1	-	ADDR(DSAR+12)
+ 5	DO	AND	BOF	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(DSAR+11)
+ 6	OB+	OR	QOF	*	*	*	*	*	*	*	IRKJ	INB	CJP	SGN	-	ADDR(DSAR+9)
+ 7	OA	OR	LAQ	*	SZRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSAR+7)
+ 8	OQ+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	CJP	YES	-	ADDR(DSAR+11)
+ 9	OA	OR	RAQ	*	SOBO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSAR+9)
+10	OQ+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+11	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+12	OA+	OR	OF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+13	OA	OR	OF	*	*	*	*	*	LST	*	*	*	*	*	*	
+14	DO	AND	BOF	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DSCR+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	IRKJ	INA	*	-	-	uDL(0020)
+ 1	OB	OR	OF	*	*	*	*	*	OIC	MRRQ	*	INB	LDCT	*	-	ADDR(IF+0)
+ 2	AQ	S-R	OF	CONE	*	*	*	INW	*	*	*	*	CJP	F0=0	-	ADDR(DSCR+4)
+ 3	OB	S-R	OF	CONE	*	*	*	*	ISGN	*	*	*	*	*	*	
+ 4	OB	OR	OF	*	*	*	OTOD	INN	*	*	IRJK	INA	CJP	W0=1	-	ADDR(DSCR+12)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DSCR+ 5	DO	AND	BOF	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(DSCR+11)
+ 6	OB+	OR	QOF	*	*	*	*	*	*	*	IRKJ	INB	CJP	SGN	-	ADDR(DSCR+9)
+ 7	OA	OR	LAQ	*	SRO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSCR+7)
+ 8	OQ+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	CJP	YES	-	ADDR(DSCR+11)
+ 9	OA	OR	RAQ	*	SQO	*	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(DSCR+9)
+10	OQ+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+11	OA	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+12	OA+	OR	OF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+13	OA	OR	OF	*	*	*	*	*	LST	*	*	*	*	*	*	
+14	DO	AND	BOF	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
SBR+ 0	DO	OR	QOF	*	*	*	BMO	*	OIC	MRRQ	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 1	AQ	OR	BOF	*	*	*	*	*	*	*	IRJK	INB	JRP	≠INT	-	ADDR(INTR+0)
SB+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	OR	QOF	*	*	*	BMO	*	OEAR	MRRQ	IRKJ	INA	*	*	*	
+ 2	DQ	OR	OF	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(SB+2)
+ 3	*	*	*	*	*	*	OUTW	*	OEAR	MWRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(SB+4)
+ 5	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
SBI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	OR	QOF	*	*	*	BMO	*	OEAR	MRRQ	IRKJ	INA	CJP	YES	-	ADDR(SB+2)
RBR+ 0	DO	XNOR	QOF	*	*	*	BMO	*	OIC	MRRQ	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 1	AQ	AND	OF	*	*	*	*	*	*	*	IRJK	INB	JRP	≠INT	-	ADDR(INTR+0)
RB+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	XNOR	QOF	*	*	*	BMO	*	OEAR	MRRQ	IRKJ	INA	*	*	*	
+ 2	DQ	AND	OF	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(RB+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
RB+ 3	*	*	*	*	*	*	OUTW	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(SB+4)
RBI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	XNOR	QOF	*	*	*	BMO	*	OEAR	MRRQ	IRKJ	INA	CJP	YES	-	ADDR(RB+2)
TBR+ 0	DO	AND	QOF	*	*	*	BMO	*	OIC	MRRQ	IRJK	INA	LDCT	*	-	ADDR(IF+0)
+ 1	AQ	AND	OF	*	*	*	*	*	LST	*	1RJK	INB	JRP	≠INT	-	ADDR(INTR+0)
TB+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	OR	QOF	*	*	*	BMO	*	OEAR	MRRQ	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	DQ	AND	OF	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(TB+2)
+ 3	DO	OR	OF	*	*	*	OUTW	*	LST	*	*	*	CJP	YES	-	ADDR(SB+5)
TBI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	OR	QOF	*	*	*	BMO	*	OEAR	MRRQ	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(TB+2)
SVBR+ 0	OA	OR	OF	*	*	*	*	INW	OIC	MRRQ	IRJK	INAB	*	*	*	
+ 1	*	*	*	*	*	*	OUTW	INIR	*	*	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	DA	OR	BOF	*	*	*	BMO	*	*	*	IRJK	*	JRP	≠INT	-	ADDR(INTR+0)
RVBR+ 0	OA	OR	OF	*	*	*	*	INW	OIC	MRRQ	IRJK	INAB	*	*	*	
+ 1	*	*	*	*	*	*	OUTW	INIR	*	*	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	DO	XNOR	QOF	*	*	*	BMO	*	*	*	IRJK	*	*	*	*	
+ 3	AQ	AND	BOF	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
IVBR+ 0	OA	OR	OF	*	*	*	*	INW	OIC	MRRQ	IRJK	INAB	*	*	*	
+ 1	*	*	*	*	*	*	OUTW	INIR	*	*	IRKJ	INA	LDCT	*	-	ADDR(IF+0)
+ 2	DA	AND	OF	*	*	*	BMO	*	*	*	IRJK	*	JRP	≠INT	-	ADDR(INTR+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
LR+ 0	OB	OR	OF	*	*	*	*	INW	OIC	MRRQ	IRJK	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	DO	OR	BOF	*	*	*	OUTW	*	LST	*	IRKJ	INB	JRP	≠INT	-	ADDR(INTR+0)
L+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	OMDR	INW	LST	*	*	*	CJP	GO=0	-	ADDR(L+2)
+ 3	DO	OR	BOF	*	*	*	OUTW	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
LI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(L+2)
LIM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	DO	OR	BOF	*	*	*	OUTW	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	OB	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
LISP+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISP+0)
+ 1	DO	OR	BOF	*	*	*	OUTW	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	OB	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
LISN+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(ISN+0)
+ 1	DO	OR	BOF	*	*	*	OUTW	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	OB	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
LB+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	*	OTOA	MRRQ	IRJK	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(22)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(L+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
LB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	*	OTOA	MRRQ	IRJK	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(22)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(L+2)
LB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	*	OTOA	MRRQ	IRJK	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(22)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(L+2)
LB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(70)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	*	OTOA	MRRQ	IRJK	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(22)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(L+2)
DLR+ 0	*	*	*	*	*	*	*	*	OIC	MRRQ	IRKJ	INAB	LDCT	*	-	ADDR(IF+0)
+ 1	OA+	OR	BOF	*	*	*	*	*	LDZO	*	*	*	*	*	*	
+ 2	OA	OR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
DL+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	*	INEA	OEAR	MRRQ	*	*	*	*	*	
+ 2	DO	OR	BOF	*	*	*	OMDR	*	LDZO	*	*	*	CJP	GO=0	-	ADDR(DL+2)
+ 3	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	DO+	OR	BOF	*	*	*	OMDR	*	LST	*	*	*	CJP	GO=0	-	ADDR(DL+4)
+ 5	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INT+0)
DLI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(DL+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
DLB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CONE	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	*	*	*	
+ 2	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	*	PLJK	INAB	*	*	*	uDS(00)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(DL+2)
DLB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CONE	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(DLB4+2)
DLB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CONE	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(DLB4+2)
DLB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(70)
+ 1	DA	ADD	OF	CONE	*	*	OJK	INEA	OTOA	MRRQ	IRJK	*	CJP	YES	-	ADDR(DLB4+2)
LUB+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	uDL	INN	CLWX	*	*	*	*	-	-	uDL(0007)
+ 3	*	*	*	*	*	*	OMDR	INX	*	*	*	*	CJP	GO=0	-	ADDR(LUB+3)
+ 4	*	*	*	*	SZRO	LW	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(LUB+4)
+ 5	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	*	*	*	
+ 6	DO	OR	BOF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
LUBI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(DAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(LUB+2)
LLB+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	uDL	INN	CLWX	*	*	*	*	-	-	uDL(0007)
+ 3	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(LLR+3)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CND	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
LLB+ 4	*	*	*	*	SZRO	RW	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(LLB+4)
+ 5	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	*	*	*	
+ 6	DO	OR	BOF	*	*	*	OUTX	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
LLBI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(LUB+2)
LLB+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	uDL	INN	CLWX	*	*	*	*	-	-	uDL(0007)
+ 3	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(LLB+3)
+ 4	*	*	*	*	SZRO	RW	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(LLB+4)
+ 5	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	*	*	*	
+ 6	DO	OR	BOF	*	*	*	OUTX	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
LLBI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(LLB+2)
LDST+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(INTR+20)
RFI+ 0	OA	OR	OF	*	*	*	*	*	OTOA	MRRQ	IRJK	INA	*	*	*	
+ 1	*	*	*	*	*	*	*	*	CEX	*	IRKJ	INB	*	*	*	
+ 2	*	*	*	*	*	*	OMDR	INIC	*	*	*	*	CJP	GO=0	-	ADDR(RFI+2)
+ 3	OA	R-S	BOF	CZRO	*	*	*	*	OTOA	MRRQ	*	*	*	*	*	
+ 4	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(RFI+4)
+ 5	OA	R-S	BOF	CZRO	*	*	OUTW	INS	OTOA	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 6	*	*	*	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0	-	ADDR(RFI+6)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CND	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
RFI+ 7	OA	R-S	BOF	CZRO	*	*	*	*	OIC	MRRQ	*	*	*	*	*	
+ 8	*	*	*	*	*	*	OUTW	*	INTE	*	*	*	JRP	YES	LMR	
LM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	*	*	*	
+ 2	DO	OR	QOF	*	*	*	OJK	*	*	*	IRKJ	*	*	*	*	
+ 3	DQ	AND	OF	*	*	*	uDL	INW	*	*	IRKJ	INB	*	-	-	uDL(000F)
+ 4	DO	AND	QOF	*	*	*	OUTW	INW	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 5	OQ	OR	OF	*	*	*	OTOD	*	*	*	DBJK	INB	CJP	N=0	-	ADDR(LM+9)
+ 6	DO	OR	BOF	*	*	*	OMDR	*	*	*	*	*	CJP	GO=0	-	ADDR(LM+6)
+ 7	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	*	*	*	
+ 8	OQ	ADD	QOF	CONE	*	*	*	*	DECN	*	*	*	CJP	YES	-	ADDR(LM+5)
+ 9	DO	ADD	BOF	*	*	*	OMDR	*	*	*	*	*	CJP	GO=0	-	ADDR(LM+9)
+10	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR)
ST+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	OA	OR	OF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(ST+3)
+ 4	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
STI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	OA	OR	OF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	CJP	YES	-	ADDR(ST+2)
STB4+ 0	OB	OR	OF	*	*	*	*	INW	*	*	PLJK	INAB	*	*	*	uDS(42)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	*	*	*	
+ 2	*	*	*	*	*	*	OUTW	*	OEAR	MWRQ	*	*	CJP	YES	-	ADDR(ST+2)
STB5+ 0	OB	OR	OF	*	*	*	*	INW	*	*	PLJK	INAB	*	*	*	uDS(52)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(STB4+2)

Table C1
Emulation Microprogram (continued)

LABEL	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
STB6+ 0	OB	OR	OF	*	*	*	*	INW	*	*	PLJK	INAB	*	*	*	uDS(62)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(STB4+2)
STB7+ 0	OB	OR	OF	*	*	*	*	INW	*	*	PLJK	INAB	*	*	*	uDS(72)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(STB4+2)
DST+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	OA	OR	OF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	*	*	*	
+ 2	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(DST+3)
+ 4	OB+	OR	OF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	*	*	*	
+ 5	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	*	
+ 6	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
DSTI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	OA	OR	OF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	CJP	YES	-	ADDR(DST+2)
DSTB4+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(40)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	*	*	*	
+ 2	OB	OR	OF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	CJP	YES	-	ADDR(DST+2)
DSTB5+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(50)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(DSTB4+2)
DSTB6+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(60)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(DSTB4+2)
DSTB7+ 0	*	*	*	*	*	*	*	*	*	*	PLJK	INAB	*	*	*	uDS(70)
+ 1	DA	ADD	OF	CZRO	*	*	OJK	INEA	*	*	IRJK	*	CJP	YES	-	ADDR(DSTB4+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
STC+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	OR	QOF	*	*	*	OJK	*	*	*	IRKJ	*	CJP	YES	-	ADDR(STCI+1)
STCI+ 0	DO	OR	QOF	*	*	*	OJK	*	*	*	IRKJ	*	CJS	YES	-	ADDR(IAM+0)
+ 1	DQ	AND	OF	*	*	*	uDL	INW	*	*	*	*	*	-	-	uDL(000F)
+ 2	*	*	*	*	*	*	OUTW	*	OEAR	MWRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(STCI+3)
+ 4	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
SRM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	OB+	OR	QOF	*	*	*	*	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	AQ	OR	OF	*	*	*	*	*	OEAR	MWRQ	*	*	*	*	*	
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(SRM+3)
+ 4	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
STUB+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	OA	OR	OF	*	*	*	*	INW	OEAR	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	SZRO	LW	OJK	INN	*	*	PLJK	*	*	*	*	uNS(02)
+ 3	*	*	*	*	SZRO	LW	*	*	DECN	*	*	*	CJP	N≠0	-	ADDR(STUB+3)
+ 4	DO	OR	QOF	*	*	*	OMDR	*	*	*	*	*	CJP	GO=0	-	ADDR(STUB+4)
+ 5	DQ	AND	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(000F)
+ 6	DQ	OR	QOF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 7	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(STUB+7)
+ 8	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
SUBI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	OA	OR	OF	*	*	*	*	INW	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(STUB+2)
STLB+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	*	*	*	

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
STLB+ 2	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(00FF)
+ 3	AQ	AND	OF	*	*	*	*	INW	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 4	DO	OR	QOF	*	*	*	OMDR	*	*	*	*	*	CJP	GO=0	-	ADDR(STLB+4)
+ 5	DQ	AND	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(FF00)
+ 6	DQ	OR	OF	*	*	*	OUTW	INW	*	*	*	*	*	*	*	
+ 7	*	*	*	*	*	*	OUTW	*	OEAR	MWRQ	*	*	*	*	*	
+ 8	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(STLB+8)
+ 9	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
SLBI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	*	*	*	*	*	*	*	*	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(STLB+2)
MOV+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	OA+	OR	OF	*	*	*	OTOD	INN	OEAR	MRRQ	*	*	*	*	*	
+ 2	DO+	AND	BOF	*	*	*	*	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	*	*	*	*	*	*	*	*	DECN	*	*	*	CJP	N=0	-	ADDR(MOV+9)
+ 4	DO	OR	OF	*	*	*	OMDR	INW	*	*	*	*	CJP	GO=0		ADDR(MOV+4)
+ 5	OA	OR	OF	*	*	*	OUTW	*	OTOA	MWRQ	*	*	*	*	*	
+ 6	OA	ADD	BOF	CONE	*	*	*	*	*	*	*	*	*	*	*	
+ 7	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(MOV+7)
+ 8	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	YES	-	ADDR(MOV+3)
+ 9	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(MOV+9)
+10	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
MOVI+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 1	OA+	OR	OF	*	*	*	OTOD	INN	*	*	*	*	CJP	YES	-	ADDR(MOV+2)
STM+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(000F)
+ 2	DQ	AND	OF	*	*	*	OJK	INW	*	*	IRKJ	*	LDCT	*	*	LDCT(STM+7)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
STM+ 3	DO	AND	QOF	*	*	*	OUTW	INN	*	*	PLJK	INAB	*	*	*	uDS(00)
+ 4	OA	OR	OF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	CJP	N=0	-	ADDR(STM+8)
+ 5	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	*	*	*	*	*	*	
+ 6	OQ	ADD	QOF	CONE	*	*	*	*	DECN	*	*	*	*	*	*	
+ 7	OQ	OR	OF	*	*	*	OTOD	*	*	*	DBJK	INA	JRP	GO=0	-	ADDR(STM+4)
+ 8	*	*	*	*	*	*	*	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 9	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(STM+9)
+10	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
PSHM+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(000F)
+ 1	DQ	AND	OF	*	*	*	OJK	ING	*	*	IRKJ	*	*	*	*	
+ 2	DQ	AND	QOF	*	*	*	OJK	*	*	*	IRJK	INB	*	*	*	
+ 3	OA	OR	OF	*	*	*	*	INEA	*	*	PLJK	INA	*	*	*	uDS(F0)
+ 4	*	*	*	*	*	*	OUTG	INN	*	*	*	*	*	*	*	
+ 5	OB	OR	OF	*	*	*	OTOD	*	OEAR	MWRQ	*	*	CJP	N=0	-	ADDR(PSHM+9)
+ 6	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	*	*	*	LDCT	*	-	ADDR(PSHM+8)
+ 7	OQ	ADD	QOF	CONE	*	*	*	*	DECN	*	*	*	*	*	*	
+ 8	OQ	OR	OF	*	*	*	OTOD	*	*	*	DBJK	INB	JRP	GO=0	-	ADDR(PSHM+5)
+ 9	DA	ADD	BOF	CONE	*	*	OUTG	*	*	*	PLJK	INAB	*	*	*	uDS(FF)
+10	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(PSHM+10)
+11	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+12	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
POPM+ 0	DO	OR	QOF	*	*	*	uDL	*	*	*	PLJK	INB	*	-	-	uDL(000F)
+ 1	OB	OR	OF	*	*	*	*	INEA	*	*	*	*	*	*	*	
+ 2	DO	S-R	OF	CZRO	*	*	ATOD	INEA	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	DQ	AND	OF	*	*	*	OJK	ING	*	*	IRKJ	*	*	*	*	
+ 4	DQ	AND	QOF	*	*	*	OJK	*	*	*	IRJK	*	*	*	*	
+ 5	DQ	ADD	QOF	CZRO	*	*	*	*	*	*	PLJK	INA	*	*	*	uDS(F0)
+ 6	*	*	*	*	*	*	OUTG	INN	*	*	*	*	*	*	*	

Table C1
Emulation Microprogram (continued)

TABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
POPM+ 7	OQ	OR	OF	*	*	*	OTOD	*	*	*	DBJK	INB	CJP	N=0	-	ADDR(POPM+1)
+ 8	DO	OR	BOF	*	*	*	OMDR	*	*	*	*	*	CJP	GO=0	-	ADDR(POPM+8)
+ 9	DO	S-R	OF	CZRO	*	*	ATOD	INEA	OEAR	MRRQ	*	*	*	*	*	
+10	OQ	R-S	QOF	CZRO	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(POPM+7)
+11	DO	OR	BOF	*	*	*	OMDR	*	*	*	*	*	CJP	GO=0	-	ADDR(POPM+11)
+12	*	*	*	*	*	*	*	*	OIC	MRRQ	PLJK	INB	*	*	*	uDS(OF)
+13	DO	OR	BOF	*	*	*	ATOD	*	OEAR	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
SJS+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	OA	ADD	BOF	CONE	*	*	OUTW	*	OTOA	MWRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	DO	OR	OF	*	*	*	ATOD	INIC	OEAR	*	*	*	*	*	*	
+ 3	*	*	*	*	*	*	*	*	CEX	*	*	*	CJP	GO=0	-	ADDR(SJS+3)
+ 4	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
URTS+ 0	OA	OR	OF	*	*	*	*	*	OTOA	MRRQ	IRJK	INA	*	*	*	
+ 1	OA	R-S	BOF	CZRO	*	*	*	*	CEX	*	IRKJ	INB	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	OMDR	INIC	*	*	*	*	CJP	GO=0	-	ADDR(URTS+2)
+ 3	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
NOP+ 0	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 1	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
BPT+ 0	*	*	*	*	*	*	*	*	BPNT	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 1	*	*	*	*	*	*	*	*	BPNT	*	*	*	CJP	GO=0	-	ADDR(BPT+3)
+ 2	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
+ 3	*	*	*	*	*	*	*	*	BPNT	*	*	*	CJP	GO=1	-	ADDR(BPT+6)
+ 4	*	*	*	*	*	*	*	*	BPNT	*	*	*	CJP	≠INT	-	ADDR(BPT+3)
+ 5	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(INTR+0)
+ 6	*	*	*	*	*	*	*	*	BPNT	*	*	*	CJP	GO=0	-	ADDR(BPT+2)
+ 7	*	*	*	*	*	*	*	*	BPNT	*	*	*	CJP	≠INT	-	ADDR(BPT+6)
+ 8	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(INTR+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CND	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
EX+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INEA	OEAR	MRRQ	*	*	CJP	EX=1	-	ADDR(EX+3)
+ 2	DO	OR	OF	*	*	*	ATOD	INH	OIC	*	*	*	*	*	*	
+ 3	DO	OR	OF	*	*	*	ATOD	INIC	OEAR	*	*	*	*	*	*	
+ 4	*	*	*	*	*	*	MDR	INIR	PCLR	*	*	*	CJP	GO=0	-	ADDR(EX+4)
+ 5	*	*	*	*	*	*	*	*	SEX	*	*	*	JMAP	YES	*	
JBC+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 1	OA	OR	OF	*	*	*	*	*	LST	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	F=0	-	ADDR(JBC+6)
+ 3	OA	R-S	*	CZRO	*	*	*	*	LST	*	*	*	*	*	*	
+ 4	DO	OR	OF	*	*	*	ATOD	INIC	OEAR	MRRQ	*	*	*	*	*	
+ 5	*	*	*	*	*	*	*	*	CEX	*	*	*	*	*	*	
+ 6	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
BEZ+ 0	DO	OR	OF	*	*	*	uDL	INW	*	*	*	*	*	-	-	uDL(0002)
+ 1	DO	OR	OF	*	*	*	uDL	INX	*	*	*	*	*	-	-	uDL(000D)
+ 2	*	*	*	*	*	*	OUTW	*	*	*	DBJK	*	CJP	≠JOC	-	ADDR(BEZ+7)
+ 3	*	*	*	*	*	*	OUTX	*	*	*	DBJK	*	CJP	JOC	-	ADDR(BEZ+7)
+ 4	DO	OR	OF	*	*	*	OJK	ING	CEX	*	IRJK	*	CJS	YES	-	ADDR(ICR+1)
+ 5	DO	OR	OF	*	*	*	ATOD	INIC	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 6	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+ 7	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 8	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
BNZ+ 0	DO	OR	OF	*	*	*	uDL	INW	*	*	*	*	*	-	-	uDL(0005)
+ 1	DO	OR	OF	*	*	*	uDL	INX	*	*	*	*	*	-	-	uDL(000A)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(BEZ+2)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
BGE+ 0	DO	OR	OF	*	*	*	uDL	INW	*	*	*	*	*	-	-	uDL(0006)
+ 1	DO	OR	OF	*	*	*	uDL	INX	*	*	*	*	*	-	-	uDL(0009)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(BEZ+2)
BR+ 0	DO	OR	OF	*	*	*	OJK	ING	CEX	*	IRJK	*	CJS	YES	-	ADDR(ICR+1)
+ 1	DO	OR	OF	*	*	*	ATOD	INIC	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
BLT+ 0	DO	OR	OF	*	*	*	uDL	INW	*	*	*	*	*	-	-	uDL(0001)
+ 1	DO	OR	OF	*	*	*	uDL	INX	*	*	*	*	*	-	-	uDL(000E)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(BEZ+2)
BLE+ 0	DO	OR	OF	*	*	*	uDL	INW	*	*	*	*	*	-	-	uDL(0003)
+ 1	DO	OR	OF	*	*	*	uDL	INX	*	*	*	*	*	-	-	uDL(000C)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(BEZ+2)
BGT+ 0	DO	OR	OF	*	*	*	uDL	INW	*	*	*	*	*	-	-	uDL(0004)
+ 1	DO	OR	OF	*	*	*	uDL	INX	*	*	*	*	*	-	-	uDL(000B)
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(BEZ+2)
JC+ 0	*	*	*	*	*	*	*	*	*	*	IRKJ	*	CJP	≠JOC	-	ADDR(JCI+4)
+ 1	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(DAM+1)
+ 2	*	*	*	*	*	*	*	*	CEX	*	*	*	CJP	YES	-	ADDR(JCI+2)
JCI+ 0	*	*	*	*	*	*	*	*	*	*	IRKJ	*	CJP	≠JOC	-	ADDR(JCI+4)
+ 1	*	*	*	*	*	*	*	*	CEX	*	*	*	CJS	YES	-	ADDR(IAM+0)
+ 2	DO	OR	OF	*	*	*	ATOD	INIC	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
+ 4	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 5	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
JS+ 0	DO	ADD	BOF	CONE	*	*	ATOD	*	OIC	MRRQ	IRKJ	INB	CJS	YES	-	ADDR(DAM+1)
+ 1	DO	OR	OF	*	*	*	ATOD	INIC	OEAR	MRRQ	*	*	LDCT	*	-	ADDR(IF+0)
+ 2	*	*	*	*	*	*	*	*	CEX	*	*	*	*	*	*	
+ 3	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
CIO+ 0	*	*	*	*	*	*	*	*	*	*	*	*	CJP	YES	-	ADDR(NOP+0)
IN+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(8000)
+ 2	DQ	AND	OF	*	*	*	OUTW	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	DQ	S-R	OF	CONE	*	*	OUTW	INEA	*	*	*	*	CJP	F=0	-	ADDR(IN+5)
+ 4	*	*	*	*	*	*	*	*	*	*	*	*	CJV	YES	*	
+ 5	*	*	*	*	*	*	*	*	OEAR	PRRQ	*	*	*	*	*	
+ 6	DO	OR	BOF	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(IN+6)
+ 7	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
ITA+ 0	DO	OR	BOF	*	*	*	TAO	*	LST	*	*	*	*	*	*	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
ITB+ 0	DO	OR	BOF	*	*	*	TBO	*	LST	*	*	*	*	*	*	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
RSW+ 0	DO	OR	BOF	*	*	*	OUTS	*	LST	*	*	*	*	*	*	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
ID+ 0	DO	OR	BOF	*	*	*	*	*	LST	DRRQ	*	*	*	*	*	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
RIC+ 0	DO	OR	BOF	*	*	*	*	*	INTE	*	*	*	*	*	RMR	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	*	*	*	
+ 2	OB	OR	OF	*	*	*	*	*	LST	*	*	*	JRP	≠INT	-	ADDR(INTR+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
OUT+ 0	DO	ADD	OF	CONE	*	*	ATOD	INIC	OIC	MRRQ	*	*	CJS	YES	-	ADDR(IMAM+1)
+ 1	DO	OR	QOF	*	*	*	uDL	*	*	*	*	*	*	-	-	uDL(4000)
+ 2	DQ	AND	OF	*	*	*	*	*	*	*	*	*	LDCT	*	-	ADDR(IF+0)
+ 3	DO	OR	OF	*	*	*	OUTW	INEA	*	*	*	*	CJP	F=0	-	ADDR(OUT+5)
+ 4	*	*	*	*	*	*	*	*	*	*	*	*	CJV	YES	*	
+ 5	OA	OR	OF	*	*	*	*	*	OEAR	PWRQ	*	*	*	*	*	
+ 6	*	*	*	*	*	*	*	*	*	*	*	*	CJP	GO=0	-	ADDR(OUT+6)
+ 7	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
GO+ 0	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	*	*	*	
+ 1	*	*	*	*	*	*	*	*	TGNG	*	*	*	JRP	≠INT	-	ADDR(INTR+0)
OTA+ 0	OA	OR	OF	*	*	*	OTOD	*	TAIN	*	*	*	*	*	*	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
OTB+ 0	OA	OR	OF	*	*	*	OTOD	*	TBIN	*	*	*	CJP	YES	-	ADDR(OTA+1)
DMAE+ 0	*	*	*	*	*	*	*	*	DMAE	*	*	*	CJP	YES	-	ADDR(OTA+1)
DMAD+ 0	*	*	*	*	*	*	*	*	DMAD	*	*	*	CJP	YES	-	ADDR(OTA+1)
DSBL+ 0	*	*	*	*	*	*	OJK	*	INTE	*	PLJK	*	*	*	*	uDS(OD)
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(SIC+2)
ENBL+ 0	*	*	*	*	*	*	OJK	*	INTE	*	PLJK	*	*	*	LSR	uDS(00)
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	EX=0	-	ADDR(IF+0)
+ 2	DO	OR	OF	*	*	*	OUTH	INIC	CEX	*	*	*	CJP	GO=0	-	ADDR(ENBL+2)
+ 3	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(IF+0)

Table C1
Emulation Microprogram (continued)

LABLE	SRCE	FUNC	DEST	CIN CNTL	SHFT MUX	EXT SHFT	REG OUT	REG IN	CMD	I BUS	JK FLD	AB IN	NXT ADDR	COND TEST	INT CNTL	DATA/ADDR
OD+ 0	OA	OR	OF	*	*	*	*	*	*	DWRQ	*	*	*	*	*	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	JRP	≠INT	-	ADDR(INTR+0)
CLIR+ 0	*	*	*	*	*	*	*	*	INTE	*	*	*	*	*	CAI	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	CJP	YES	-	ADDR(SIC+2)
SIC+ 0	OA	OR	OF	*	*	*	OTOD	*	INTE	*	*	*	*	*	SMR	
+ 1	*	*	*	*	*	*	*	*	OIC	MRRQ	*	*	*	*	*	
+ 2	*	*	*	*	*	*	*	*	*	*	*	*	JRP	≠INT	-	ADDR(INTR+0)

VITA

Frederick George Thourot was born on 8 September 1948 at Oakland, California. He graduated from Tucson High School, Tucson, Arizona, in 1967 and entered the University of Arizona at Tucson where he participated in the AFROTC program. In 1971 he graduated from the University of Arizona, receiving a Bachelor of Science degree in Electrical Engineering, and was commissioned in the US Air Force.

Upon entry into active duty in the US Air Force in September 1971, he was assigned to the Communications-Electronic Engineer School at Keesler AFB, Mississippi. Between May 1972 and May 1976 he was assigned to the Headquarters Pacific Communications Area (AFCS) as an electronic engineer. In June 1976 he entered the Air Force Institute of Technology. He is a member of Tau Beta Pi and Eta Kapa Nu.

Permanent Address: 2524 N. Goyette

Tucson, Arizona 85712

This thesis was typed by Nalda Blair.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

14 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/77-10	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN OF THE PROCESSOR FOR SOFTWARE COMPATIBLE AVIONIC COMPUTER FAMILY		5. TYPE OF REPORT & PERIOD COVERED<br~~MS Thesis~~ Master's thesis. 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Frederick G. Thourot<br~~Captain USAF~~		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Avionics Laboratory (AFAL/AAT) Wright-Patterson AFB, Ohio 45433		12. REPORT DATE Dec[redacted]77 13. NUMBER OF PAGES 232
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; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

JERRAL F. GUESS, Captain, USAF
Director of Information

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

Software Compatible Avionic Computer Family; Processor Design; Microprocessor; AM 2900 Applications

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

This report presents the design of a microprogrammable processor which emulates the baseline instruction set of the Software-Compatible Avionic Computer Family. The general architecture of this processor includes 16/32 bit word lengths, hardware fixed point arithmetic, firmware floating point arithmetic, and hardware vectored interrupts. The processors uses a flexible internal bus (I-BUS) to interface with memory modules, programmed input/output channels, and DMA channels. Integrated circuit devices from the AM 2900 family are used to realize the design. The processor's arithmetic and logic unit contains four

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

012 225

AM 2901A bit slice devices which are cascaded by an AM 2902 carry look-ahead generator. The processor's control unit uses an AM 2910 microsequencer to address control memory in a first level pipeline mode. The processor's hardware vectored interrupt system is managed by two AM 2914 priority interrupt encoders. Evaluation of the processor design determined that the processor's operational speed falls below the design goal of 200 to 500 KOPS. The report concludes with recommendations for improving the processor's speed by adding hardware to facilitate floating point arithmetic and to pipeline I/O transfers over the I-BUS.