

AD-A100 577

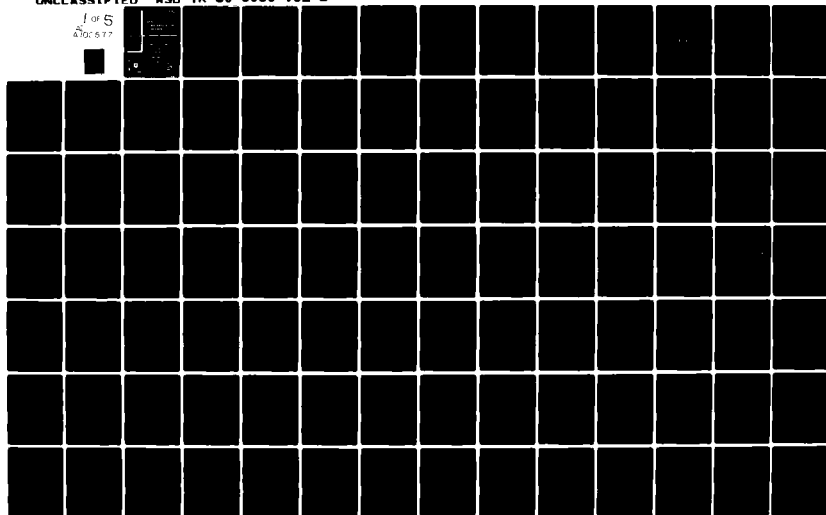
AERONAUTICAL SYSTEMS DIV WRIGHT-PATTERSON AFB OH
AFSC STANDARDIZATION CONFERENCE, 1553, 1589, 1750, 1760, ADA, N--ETC(U)
NOV 80 E C GANGL, S E SMITH
ASD-TR-80-5050-VOL-2

F/G 1/3

UNCLASSIFIED

NL

for 5
AD-A100 577



AD A100577

14

ASD-TR-80-5050

VOL-2

6

LEVEL III

AFSC

STANDARDIZATION CONFERENCE

1553, 1589, 1750, 1760, Ada,



NOVEMBER 18 - 20, 1980

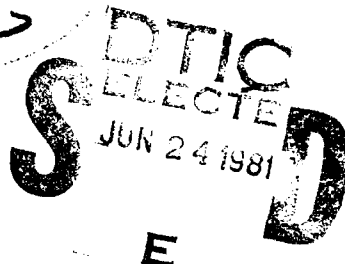
11 Nov 84

12478

DAYTON CONVENTION CENTER

DAYTON, OHIO

**Volume II.
PROCEEDINGS**



E

HOSTED BY:



AERONAUTICAL

SYSTEMS DIVISION

SPONSORED BY:



AIR FORCE

SYSTEMS COMMAND

VOL II OF II

STANDARDS

**APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED**

008800 81 6 24 282 LB

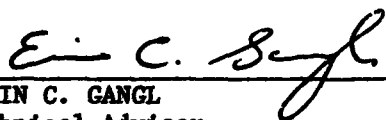
DTIC FILE COPY

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.



ERWIN C. GANGL
Technical Advisor
Information Engineering Division



RONALD LONGBRAKE
Technical Director (Actg)
Directorate of Avionics Engineering



ROBERT P. LAVOIE, Colonel, USAF
Director, Avionics Engineering
Deputy for Engineering

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization, please notify ASD/ENAI, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ASD-TR-80-5050	2. GOVT ACCESSION NO. AD-A100 577	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Proceedings - Standards of the Avionics Standardization Conference		5. TYPE OF REPORT & PERIOD COVERED Vol II of II 18-20 November 1980
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Editors: Erwin C. Gangl Stephen E. Smith		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Hq ASD/ENAI Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Hq ASD/ENA Wright-Patterson AFB, Ohio 45433		12. REPORT DATE 18-20 November 1980
		13. NUMBER OF PAGES 477
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same as above.		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) N/A		
18. SUPPLEMENTARY NOTES N/A		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer Instruction Set Architecture, Multiplexing, Compilers, Support Soft- ware, Data Bus, MIL-STD-1589, MIL-STD-1750, MIL-STD-1553, MIL-STD-1760, Digital Avionics, Systems Integration, Stores Interface, Standardization.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This is a collection of UNCLASSIFIED mil-standards to be distributed at the AFSC Avionics Standardization Conference at the Convention Center, Dayton, Ohio. The purpose of the conference is to state AF policy on standardization, educate government and industry management, and present and demonstrate hard- ware and software tools.		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ASD-TR-80-5050
Vol II of II

PROCEEDINGS, STANDARDS
OF THE
AFSC STANDARDIZATION CONFERENCE

18-20 NOVEMBER 1980

DAYTON CONVENTION CENTER
DAYTON, OHIO

Accession For	
NTIS GPO&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

RE: ASD-TR-80-5050, Vol. II
Use title on front cover per Mrs. Marie
Jankovich, ASD/ENIA

Approved for public release;
distribution unlimited.

Sponsored by: AIR FORCE SYSTEMS COMMAND

Hosted by: AERONAUTICAL SYSTEMS DIVISION

Organized by: DIRECTORATE OF AVIONICS ENGINEERING (ASD/ENA)
AERONAUTICAL SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

FOREWORD

Air Force Systems Command sponsored two MIL-STD-1553 Multiplex Data Bus Conferences, one in 1976 and one in 1978. In the past two years, MIL-STD-1750 and MIL-STD-1589 have matured to the point of justifying another conference. For the next three days we will provide an update of MIL-STD-1553B, 1750A, 1589B, and also the status of the new MIL-STD-1760, and the progress of the Ada programming language. Significant interest in MIL-STD-1553B systems applications, LSI developments and test equipment, as well as the publication of MIL-STD-1750A and 1589B have prompted this Avionics Standardization Conference. Brig Gen Jacobs, Deputy Chief of Staff, Plans and Programs, requested this conference, being hosted by the Aeronautical Systems Division, Lt Gen Skantze, Commander.

The purpose of the conference is to present Air Force standardization policy and the status of the military standards, to exchange data on lessons learned from system applications, and to present new hardware/software developments in support of these standards.

This is the STANDARDS Volume, Volume II of the Standardization Conference. Volume I will contain the proceedings and papers given during the conference, which will be sent to attendees at a later date.

Many thanks to Captain David Herrelko (AFSC/XR) and Major Al Kopp (AFSC/XRF) for their headquarter's assistance in organizing this conference; also to the ASD/ENA staff who assisted in organizing the technical program, proceedings and exhibits, Mr. Stephen Smith and Mr. Erwin Gangl. Special thanks to Mr. Joseph Militello (University of Dayton Research Institute) who did an outstanding job in organizing the conference facilities, hotel accommodations and food service.

Thanks also to the moderators and all the speakers who responded with outstanding presentations and papers in a timely manner despite such short notice. And, finally to the secretaries, Mrs. Marie Jankovich, Mrs. Kathy Hayes, and Mrs. Sharleen Thompson for their expert administrative help in handling the conference correspondence and registration.


ROBERT P. LAVOIE, Colonel, USAF
Director, Avionics Engineering
Deputy for Engineering

ACTION	1
	2
	3
	4
	5
	6
	7
	8
	9
	10
	11
	12

m. c. 28
 (m. c.)
 AXA
 AIT
 J. R.

REPLY TO
 ATTENTION XR

DEPARTMENT OF THE AIR FORCE
 HEADQUARTERS AIR FORCE SYSTEMS COMMAND
 ANDREWS AIR FORCE BASE, DC 20334



29 APR 1980

SUBJECT MIL-STD-1750(USAF) Instruction Set Architecture Conference

TO ASD/CV

1. We are impressed with your significant technological progress in developing and implementing our Airborne Computer Instruction Set Architecture, MIL-STD-1750(USAF). The teamwork of ASD/AX, ASD/EN and the AFWAL/AA has been a key factor in this successful effort. The enthusiastic industry-wide response to this standardization initiative attests to the soundness of the approach. With over a dozen contractors implementing the 1750 architecture, and with the strong interest being shown by systems houses and airframe manufacturers, the time seems right for a conference devoted to MIL-STD-1750.
2. We request you host such a conference, in conjunction with the Fall meetings of the MIL-STD-1750 Users Group and the MIL-STD-1750 Control Board, which we understand will be held at Wright-Patterson Air Force Base. This seems an excellent time to demonstrate actual working hardware, to exchange information through presentations and a published proceedings of the conference, and to show off EN's SEAFAC facility, which has a central role in MIL-STD-1750 verification.
3. The AFSC project officer for this effort, Captain David A. Herrelko (AFSC/XRF), AUTOVON 858-5731, is prepared to provide you with any assistance you might need in arranging this conference.

FOR THE COMMANDER

Robert H. J. J. J.

Cy to: ASD/AX
 ASD/EN
 AFWAL/AA
 AFSC/DL

1. J. R. J. J. J. Gen, USAF
 2. J. R. J. J. J. and Programs

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AERONAUTICAL SYSTEMS DIVISION (AFSC)
WRIGHT-PATTERSON AIR FORCE BASE OHIO 45433



REF: TO
ATTN OF: EN

SUBJECT: AFSC Standardization Conference

TO: AFSC/XR

1. Reference your letter of 29 April 1980, Subject: MIL-STD-1750(USAF) Instruction Set Architecture Conference. ASD/EN took on the task to support the MIL-STD-1750 conference. We had originally planned to schedule this conference for mid-September. For a variety of reasons, we elected to expand the scope of this conference to cover all current AFSC standardization initiatives and issues. We have discussed this with ASD/AX and ASD/XR and have jointly agreed to separate the 1750 Technical Users' Group meeting from the broader aspects of the conference. We have tentatively rescheduled it for the period of 18 through 20 November and plan to include MIL-STDs-1750A, -1589, -1553B, and -1760.

2. We believe it would be more effective to target such a conference to SPO level managers and engineers and their counterparts in industry, in order to facilitate application of these standards to specific programs. We want to make it an educational rather than a technical symposium. This will afford us the opportunity to present our perception of the benefits we expect from incorporating the standards; present the status of key efforts supporting each of these; and present lessons learned to date where we have had some actual experience in applying them. We will also plan to address DOD standardization policy where it applies and will provide the opportunity for Headquarters personnel to make inputs addressing specific USAF and AFSC intentions. We also intend to provide our people some insight into foreign applications of our military standards.

3. I would appreciate your support of this conference and your concurrence that this approach will satisfy your original request. The Directorate of Avionics Engineering, ASD/ENA, will run the show for ASD. Our project officers are Mr. Erwin C. Gangl, ASD/ENAI, Autovon 785-4865 and Mr. Stephen Smith, ASD/ENAIB, Autovon 785-2248. They are prepared to provide your staff with any information they may need or wish to have relative to the details of the conference.

FORWARDED TO ENA
DATE 11/1/80
BY 11/1/80

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AERONAUTICAL SYSTEMS DIVISION (AFSC)
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433



Office of the Commander

17 October 1980

SUBJECT: AFSC Avionics Standardization Conference, 18 - 20 November 1980

TO: CONFERENCE ATTENDEES

1. The Aeronautical Systems Division is organizing and hosting a conference on Avionics Standardization at the Dayton Convention Center, 18 - 20 November 1980. This conference will address Air Force Avionics Standardization Policy, the status of MIL-STD-1553 (Multiplexed Data Bus), MIL-STD-1750 (Computer Instruction Set Architecture), MIL-STD-1582 (J73 Higher Order Language), MIL-STD-1760 (Aircraft to Stores Electrical Interface), and the progress of the new international Higher Order Language, Ada.
2. We have targeted this conference to managers and engineers in the Systems Program Offices and aerospace industry to facilitate the application of these standards to Air Force programs. Request you encourage attendance of your key personnel to increase their understanding and appreciation of the benefits to be gained from applying these standards.
3. We plan to make it an educational rather than technical symposium. This will afford us the opportunity to present our perception of the benefits we expect from incorporating the standards, present the status of key efforts supporting each of these, and present lessons learned to date where we have had some actual experience in applying them. We will also plan to address POD standardization policy where it applies, and will provide the opportunity for Headquarters personnel to make inputs addressing specific USAF and AFSC intentions. In addition, we will provide attendees some insight into foreign acceptance and system applications of these military standards.
4. A minimal subsistence fee of \$3.00 per person will be charged to maximize attendance. Registration will be at the door and attendees will receive a complimentary copy of the proceedings.

JAMES A. STALITZ
Major General, USAF
Commander

1 Atch
Conference Agenda

PROCEEDINGS

STANDARDIZATION CONFERENCE

18-20 NOVEMBER 1980

DAYTON, OHIO

	<u>Page</u>
MIL-STD-1553B, 21 Sept 78, Aircraft Internal Time Division Command/Response Multiplex Data Bus	1
MIL-STD-1589B, 6 June 80, JOVIAL (J73)	45
MIL-STD-1750A, 2 July 80, Sixteen-Bit Computer Instruction Set Architecture	223
MIL-STD-1760 (DRAFT), May 80, Standard Store Interface, Aircraft/Stores Electrical Interface Definition	381
Ada Programming Language	471

MIL-STD-1553B
21 September 1978
SUPERSEDING
MIL-STD-1553A
30 April 1975

MILITARY STANDARD

AIRCRAFT INTERNAL TIME DIVISION
COMMAND/RESPONSE MULTIPLEX DATA BUS



MIL-STD-1553B
21 September 1978

DEPARTMENT OF DEFENSE
Washington, D.C. 20360

Aircraft Internal Time Division Command/Response Multiplex Data Bus

MIL-STD-1553B

1. This Military Standard is approved for use by all Department and Agencies of the Department of Defense.
2. Beneficial comments (recommendations, additions, deletions) and any pertinent data which may be of use in improving this document should be addressed to: Aeronautical Systems Division, Attn: ENA1, Wright-Patterson Air Force Base 45433, by using the self addressed Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document or by letter.

FOREWORD

This standard contains requirements for aircraft internal time division command response multiplex data bus techniques which will be utilized in systems integration of aircraft subsystems. Even with the use of this standard, subtle differences will exist between multiplex data buses used on different aircraft due to particular aircraft mission requirements and the designer options allowed in this standard. The system designer must recognize this fact and design the multiplex bus controller hardware and software to accommodate such differences. These designer selected options must exist, so as to allow the necessary flexibility in the design of specific multiplex systems in order to provide for the control mechanism, architecture redundancy, degradation concept and traffic patterns peculiar to the specific aircraft mission requirements.

CONTENTS

<u>Paragraph</u>		<u>Page</u>
1.	SCOPE	1
1.1	Scope	1
1.2	Application	1
2.	REFERENCED DOCUMENTS	1
2.1	Issue of Document	1
3.	DEFINITIONS	1
3.1	Bit	1
3.2	Bit Rate	1
3.3	Pulse Code Modulation (PCM)	1
3.4	Time Division Multiplexing (TDM)	1
3.5	Half Duplex	1
3.6	Word	1
3.7	Message	3
3.8	Subsystem	3
3.9	Data Bus	3
3.10	Terminal	3
3.11	Bus Controller	3
3.12	Bus Monitor	3
3.13	Remote Terminal (RT)	3
3.14	Asynchronous Operation	3
3.15	Dynamic Bus Control	3
3.16	Command/Response	3
3.17	Redundant Data Bus	3
3.18	Broadcast	3
3.19	Mode Code	3
4.	GENERAL REQUIREMENTS	4
4.1	Test and Operating Requirements	4
4.2	Data Bus Operation	4
4.3	Characteristics	4
4.3.1	Data Form	4
4.3.2	Bit Priority	4
4.3.3	Transmission Method	4
4.3.3.1	Modulation	4
4.3.3.2	Data Code	4
4.3.3.3	Transmission Bit Rate	4
4.3.3.4	Word Size	4
4.3.3.5	Word Formats	4
4.3.3.5.1	Command Word	8
4.3.3.5.1.1	Sync	8
4.3.3.5.1.2	Remote Terminal Address	8
4.3.3.5.1.3	Transmit/Receive	8
4.3.3.5.1.4	Subaddress/Mode	8
4.3.3.5.1.5	Data word Count/Mode Code	8
4.3.3.5.1.6	Parity	8
4.3.3.5.1.7	Optional Mode Control	8
4.3.3.5.1.7.1	Dynamic bus Control	9

CONTENTS (cont'd)

<u>Paragraph</u>		<u>Page</u>
4.3.3.5.1.7.2	Synchronize (Without Data Word)	9
4.3.3.5.1.7.3	Transmit Status Word	9
4.3.3.5.1.7.4	Initiate Self Test	9
4.3.3.5.1.7.5	Transmitter Shutdown	9
4.3.3.5.1.7.6	Override Transmitter Shutdown	9
4.3.3.5.1.7.7	Inhibit T/F Bit	9
4.3.3.5.1.7.8	Override Inhibit T/F Bit	9
4.3.3.5.1.7.9	Reset Remote Terminal	9
4.3.3.5.1.7.10	Reserved Mode Codes (01001 to 01111)	9
4.3.3.5.1.7.11	Transmit Vector Word	11
4.3.3.5.1.7.12	Synchronize (With Data Word)	11
4.3.3.5.1.7.13	Transmit Last Command Word	11
4.3.3.5.1.7.14	Transmit Built-In-Test (BIT) Word	11
4.3.3.5.1.7.15	Selected Transmitter Shutdown	11
4.3.3.5.1.7.16	Override Selected Transmitter Shutdown	11
4.3.3.5.1.7.17	Reserved mode Codes (10110 to 11111)	11
4.3.3.5.2	Data Word	11
4.3.3.5.2.1	Sync	11
4.3.3.5.2.2	Data	12
4.3.3.5.2.3	Parity	12
4.3.3.5.3	Status Word	12
4.3.3.5.3.1	Sync	12
4.3.3.5.3.2	RT Address	12
4.3.3.5.3.3	Message Error Bit	12
4.3.3.5.3.4	Instrumentation Bit	12
4.3.3.5.3.5	Service Request Bit	12
4.3.3.5.3.6	Reserved Status Bits	12
4.3.3.5.3.7	Broadcast Command Received Bit	13
4.3.3.5.3.8	Busy Bit	13
4.3.3.5.3.9	Subsystem Flag Bit	13
4.3.3.5.3.10	Dynamic Bus Control Acceptance Bit	13
4.3.3.5.3.11	Terminal Flag Bit	13
4.3.3.5.3.12	Parity Bit	13
4.3.3.5.4	Status Word Reset	13
4.3.3.6	Message Formats	13
4.3.3.6.1	Bus controller to Remote Terminal Transfers	14
4.3.3.6.2	Remote Terminal to Bus Controller Transfers	14
4.3.3.6.3	Remote Terminal to Remote Terminal Transfers	14
4.3.3.6.4	Mode Command Without Data Word	14
4.3.3.6.5	Mode Command With Data Word (Transmit)	14
4.3.3.6.6	Mode Command With Data Word (Receive)	14
4.3.3.6.7	optional Broadcast Command	14
4.3.3.6.7.1	Bus controller to Remote Terminal(s) Transfer (Broadcast)	14
4.3.3.6.7.2	Remote Terminal to Remote Terminal(s) Transfer (Broadcast)	17
4.3.3.6.7.3	Mode Command Without Data Word (Broadcast)	17
4.3.3.6.7.4	Mode Command With Data Word (Broadcast)	17
4.3.3.7	Intermessage Gap	17
4.3.3.8	Response Time	17

CONTENTS (Cont'd)

<u>Paragraph</u>		<u>Page</u>
4.3.3.9	Minimum No-Response Time-Out	17
4.4	Terminal Operation	17
4.4.1	Common Operation	17
4.4.1.1	Word Validation	21
4.4.1.2	Transmission Continuity	21
4.4.1.3	Terminal Fail-Safe	21
4.4.2	Bus Controller Operation	21
4.4.3	Remote Terminal	21
4.4.3.1	Operation	21
4.4.3.2	Superseding Valid Commands	21
4.4.3.3	Invalid Commands	21
4.4.3.4	Illegal Command	21
4.4.3.5	Valid Data Reception	22
4.4.3.6	Invalid Data Reception	22
4.4.4	Bus Monitor Operation	22
4.5	Hardware Characteristics	22
4.5.1	Data Bus Characteristics	22
4.5.1.1	Cable	22
4.5.1.2	Characteristics Impedance	22
4.5.1.3	Cable Attenuation	22
4.5.1.4	Cable Termination	22
4.5.1.5	Cable Stub Requirements	22
4.5.1.5.1	Transformer Coupled Stubs	23
4.5.1.5.1.1	Coupling Transformer	23
4.5.1.5.1.1.1	Transformer Input Impedance	23
4.5.1.5.1.1.2	Transformer Waveform Integrity	23
4.5.1.5.1.1.3	Transformer Common Mode Rejection	23
4.5.1.5.1.2	Fault Isolation	23
4.5.1.5.1.3	Cable Coupling	23
4.5.1.5.1.4	Stub Voltage Requirements	23
4.5.1.5.2	Direct Coupled Stubs	23
4.5.1.5.2.1	Fault Isolation	23
4.5.1.5.2.2	Cable Coupling	25
4.5.1.5.2.3	Stub Voltage Requirements	25
4.5.1.5.3	Wiring and Cabling for EMC	25
4.5.2	Terminal Characteristics	25
4.5.2.1	Terminals With Transformer Coupled Stubs	25
4.5.2.1.1	Terminal Output Characteristics	25
4.5.2.1.1.1	Output Levels	25
4.5.2.1.1.2	Output waveform	25
4.5.2.1.1.3	Output Noise	25
4.5.2.1.1.4	Output Symmetry	25
4.5.2.1.2	Terminal Input Characteristics	25
4.5.2.1.2.1	Input waveform Compatibility	27
4.5.2.1.2.2	Common Mode Rejection	27
4.5.2.1.2.3	Input Impedance	27
4.5.2.1.2.4	Noise Rejection	27
4.5.2.2	Terminals With Direct Coupled Stubs	27
4.5.2.2.1	Terminal Output Characteristics	27
4.5.2.2.1.1	Output Levels	27
4.5.2.2.1.2	Output waveform	29

CONTENTS (Cont'd)

<u>Paragraph</u>		<u>Page</u>
4.5.2.2.1.3	Output Noise	29
4.5.2.2.1.4	Output Symmetry	29
4.5.2.2.2	Terminal Input Characteristics	29
4.5.2.2.2.1	Input Waveform Compatibility	29
4.5.2.2.2.2	Common Mode Rejection	29
4.5.2.2.2.3	Input Impedance	29
4.5.2.2.2.4	Noise Rejection	30
4.6	Redundant Data Bus Requirements	30
4.6.1	Electrical Isolation	30
4.6.2	Single Event Failures	30
4.6.3	Dual Standby Redundant Data Bus	30
4.6.3.1	Data Bus Activity	30
4.6.3.2	Reset Data Bus Transmitter	30
5.	DETAIL REQUIREMENTS	30

Paragraph

Page

FIGURES

1	Sample Multiplex Data Bus Architecture	2
2	Data Encoding	5
3	Word Formats	6
4	Command and Status Sync	7
5	Data Sync	7
6	Information Transfer Formats	15
7	Broadcast Information Transfer Formats	16
8	Intermessage Gap and Response Time	18
9	Data Bus Interface Using Trans. Coupling	19
10	Data Bus Interface Using Direct Coupling	20
11	Coupling Transformer	24
12	Terminal I/O Characteristics for Transformer Coupled and Direct Coupled Stubs	24
13	Output Waveform	26

TABLES

I	Assigned Mode Codes	10
II	Criteria for Acceptance or Rejection of a Terminal for the Noise Rejection Test	28

APPENDIX

10	General	31
10.1	Redundancy	31
10.2	Bus Controller	31
10.3	Multiplex Selection Criteria	33
10.4	High Reliability Requirements	33
10.5	Stubbing	33
10.6	Use of Broadcast Option	34

APPENDIX FIGURES

10.1	Illustration of Possible Redundancy	32
10.2	Illustration of Possible Redundancy	32

1. SCOPE

1.1 Scope. This standard establishes requirements for digital, command/response, time division multiplexing (Data Bus) techniques on aircraft. It encompasses the data bus line and its interface electronics illustrated on figure 1, and also defines the concept of operation and information flow on the multiplex data bus and the electrical and functional formats to be employed.

1.2 Application. When invoked in a specification or statement of work, these requirements shall apply to the multiplex data bus and associated equipment which is developed either alone or as a portion of an aircraft weapon system or subsystem development. The contractor is responsible for invoking all the applicable requirements of this Military Standard on any and all subcontractors he may employ.

2. REFERENCED DOCUMENTS

2.1 Issue of document. The following document, of the issue in effect on date of invitation for bid or request for proposal, forms a part of the standard to the extent specified herein.

SPECIFICATION

MILITARY

MIL-E-6051 Electromagnetic Compatibility Requirements, Systems

(Copies of specifications, standards, drawings, and publications required by contractors in connection with specific procurement functions should be obtained from the procuring activity or as directed by the contracting officer.)

3. DEFINITIONS

3.1 Bit. Contraction of binary digit: may be either zero or one. In information theory a binary digit is equal to one binary decision or the designation of one of two possible values or states of anything used to store or convey information.

3.2 Bit rate. The number of bits transmitted per second.

3.3 Pulse code modulation (PCM). The form of modulation in which the modulation signal is sampled, quantized, and coded so that each element of information consists of different types or numbers of pulses and spaces.

3.4 Time division multiplexing (TDM). The transmission of information from several signal sources through one communication system with different signal samples staggered in time to form a composite pulse train.

3.5 Half duplex. Operation of a data transfer system in either direction over a single line, but not in both directions on that line simultaneously.

3.6 Word. In this document a word is a sequence of 16 bits plus sync and parity. There are three types of words: command, status and data.

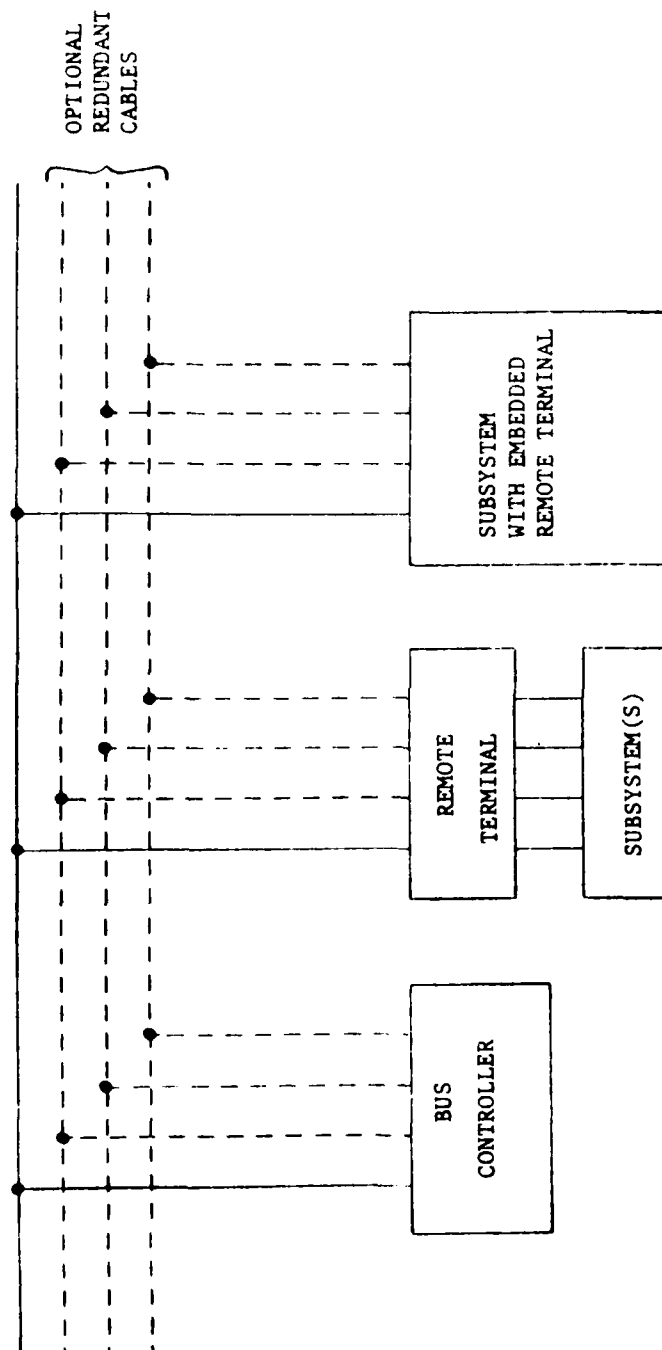


FIGURE 1. Sample multiplex data bus architecture.

- 3.7 Message. A single message is the transmission of a command word, status word, and data words if they are specified. For the case of a remote terminal to remote terminal (RT to RT) transmission, the message shall include the two command words, the two status words, and data words.
- 3.8 Subsystem. The device or functional unit receiving data transfer service from the data bus.
- 3.9 Data bus. Whenever a data bus or bus is referred to in this document it shall imply all the hardware including twisted shielded pair cables, isolation resistors, transformers, etc., required to provide a single data path between the bus controller and all the associated remote terminals.
- 3.10 Terminal. The electronic module necessary to interface the data bus with the subsystem and the subsystem with the data bus. Terminals may exist as separate line replaceable units (LRU's) or be contained within the elements of the subsystem.
- 3.11 Bus controller. The terminal assigned the task of initiating information transfers on the data bus.
- 3.12 Bus monitor. The terminal assigned the task of receiving bus traffic and extracting selected information to be used at a later time.
- 3.13 Remote terminal (RT). All terminals not operating as the bus controller or as a bus monitor.
- 3.14 Asynchronous operation. For the purpose of this standard, asynchronous operation is the use of an independent clock source in each terminal for message transmission. Decoding is achieved in receiving terminals using clock information derived from the message.
- 3.15 Dynamic bus control. The operation of a data bus system in which designated terminals are offered control of the data bus.
- 3.16 Command/Response. Operation of a data bus system such that remote terminals receive and transmit data only when commanded to do so by the bus controller.
- 3.17 Redundant data bus. The use of more than one data bus to provide more than one data path between the subsystems, i.e., dual redundant data bus, tri-redundant data bus, etc.
- 3.18 Broadcast. Operation of a data bus system such that information transmitted by the bus controller or a remote terminal is addressed to more than one of the remote terminals connected to the data bus.
- 3.19 Mode code. A means by which the bus controller can communicate with the multiplex bus related hardware, in order to assist in the management of information flow.

4. GENERAL REQUIREMENTS

4.1 Test and operating requirements. All requirements as specified herein shall be valid over the environmental conditions which the multiplex data bus system shall be required to operate.

4.2 Data bus operation. The multiplex data bus system in its most elemental configuration shall be as shown on figure 1. The multiplex data bus system shall function asynchronously in a command/response mode, and transmission shall occur in a half-duplex manner. Sole control of information transmission on the bus shall reside with the bus controller, which shall initiate all transmissions. The information flow on the data bus shall be comprised of messages which are, in turn, formed by three types of words (command, data, and status) as defined in 4.3.3.5.

4.3 Characteristics

4.3.1 Data form. Digital data may be transmitted in any desired form, provided that the chosen form shall be compatible with the message and word formats defined in this standard. Any unused bit positions in a word shall be transmitted as logic zeros.

4.3.2 Bit priority. The most significant bit shall be transmitted first with the less significant bits following in descending order of value in the data word. The number of bits required to define a quantity shall be consistent with the resolution or accuracy required. In the event that multiple precision quantities (information accuracy or resolution requiring more than 16 bits) are transmitted, the most significant bits shall be transmitted first, followed by the word(s) containing the lesser significant bits in numerical descending order. Bit packing of multiple quantities in a single data word is permitted.

4.3.3 Transmission method

4.3.3.1 Modulation. The signal shall be transferred over the data bus in serial digital pulse code modulation form.

4.3.3.2 Data code. The data code shall be Manchester II bi-phase level. A logic one shall be transmitted as a bipolar coded signal 1/0 (i.e., a positive pulse followed by a negative pulse). A logic zero shall be a bipolar coded signal 0/1 (i.e., a negative pulse followed by a positive pulse). A transition through zero occurs at the midpoint of each bit time (see figure 2).

4.3.3.3 Transmission bit rate. The transmission bit rate on the bus shall be 1.0 megabit per second with a combined accuracy and long-term stability of ± 0.1 percent (i.e., ± 1000 Hertz (Hz)). The short-term stability (i.e., stability over 1.0 second interval) shall be at least 0.01 percent (i.e., ± 100 Hz).

4.3.3.4 Word size. The word size shall be 16 bits plus the sync waveform and the parity bit for a total of 20 bits times as shown on figure 3.

4.3.3.5 Word formats. The word formats shall be as shown on figure 3 for the command, data, and status words.

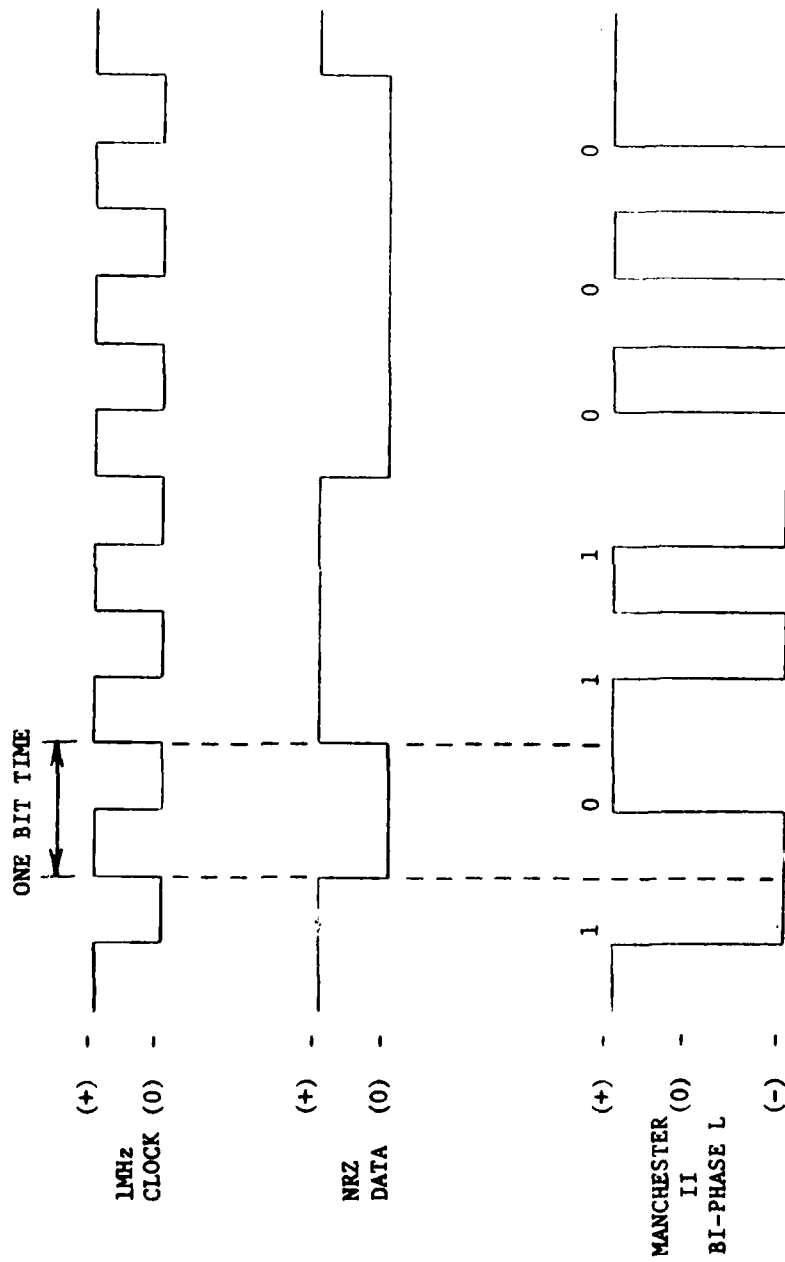


FIGURE 2. Data encoding.

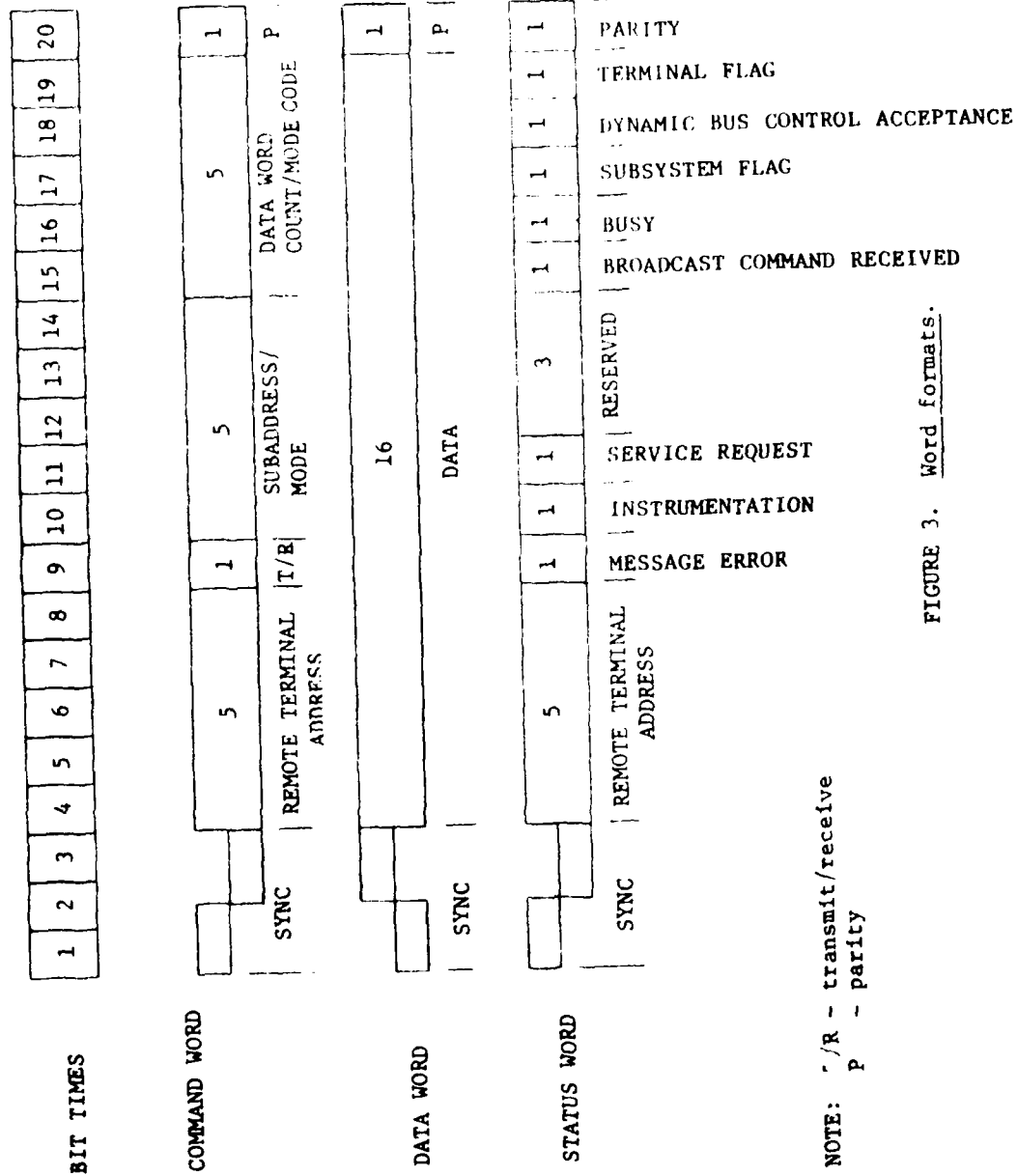


FIGURE 3. Word formats.

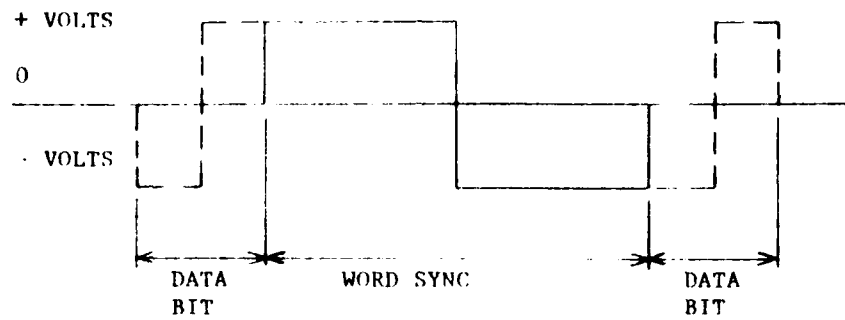


FIGURE 4. Command and status sync.

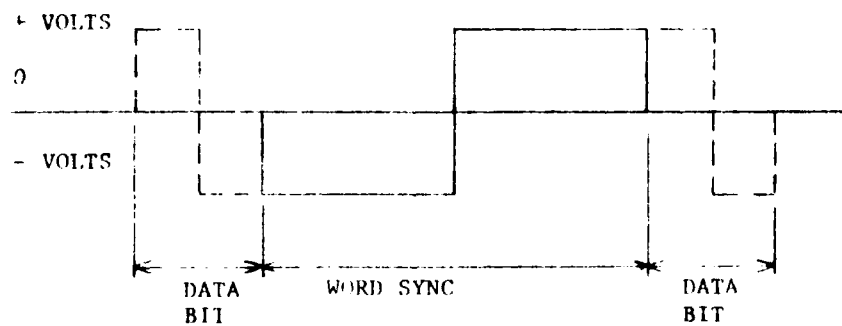


FIGURE 5. Data sync.

4.3.3.5.1 Command word. A command word shall be comprised of a sync waveform, remote terminal address field, transmit/receive (T/R) bit, subaddress/mode field, word count/mode code field, and a parity (P) bit (see figure 3).

4.3.3.5.1.1 Sync. The command sync waveform shall be an invalid Manchester waveform as shown on figure 4. The width shall be three bit times, with the sync waveform being positive for the first one and one-half bit times, and then negative for the following one and one-half bit times. If the next bit following the sync waveform is a logic zero, then the last half of the sync waveform will have an apparent width of two clock periods due to the Manchester encoding.

4.3.3.5.1.2 Remote terminal address. The next five bits following the sync shall be the RT address. Each RT shall be assigned a unique address. Decimal address 31 (11111) shall not be assigned as a unique address. In addition to its unique address, a RT shall be assigned decimal address 31 (11111) as the common address, if the broadcast option is used.

4.3.3.5.1.3 Transmit/receive. The next bit following the remote terminal address shall be the T/R bit, which shall indicate the action required of the RT. A logic zero shall indicate the RT is to receive, and a logic one shall indicate the RT is to transmit.

4.3.3.5.1.4 Subaddress/mode. The next five bits following the R/T bit shall be utilized to indicate an RT subaddress or use of mode control, as is dictated by the individual terminal requirements. The subaddress/mode values of 00000 and 11111 are reserved for special purposes, as specified in 4.3.3.5.1.7, and shall not be utilized for any other function.

4.3.3.5.1.5 Data word count/mode code. The next five bits following the subaddress/mode field shall be the quantity of data words to be either sent out or received by the RT or the optional mode code as specified in 4.3.3.5.1.7. A maximum of 32 data words may be transmitted or received in any one message block. All 1's shall indicate a decimal count of 31, and all 0's shall indicate a decimal count of 32.

4.3.3.5.1.6 Parity. The last bit in the word shall be used for parity over the preceding 16 bits. Odd parity shall be utilized.

4.3.3.5.1.7 Optional mode control. For RT's exercising this option a subaddress/mode code of 00000 or 11111 shall imply that the contents of the data word count/mode code field are to be decoded as a five bit mode command. The mode code shall only be used to communicate with the multiplex bus related hardware, and to assist in the management of information flow, and not to extract data from or feed data to a functional subsystem. Codes 00000 through 01111 shall only be used for mode codes which do not require transfer of a data word. For these codes, the T/R bit shall be set to 1. Codes 10000 through 11111 shall only be used for mode codes which require transfer of a single data word. For these mode codes, the T/R bit shall indicate the direction of data word flow as specified in 4.3.3.5.1.3. No multiple data word transfer shall be implemented with any mode code. The mode codes are reserved for the specific functions as specified in table I and shall not be used for any other purpose. If the designer chooses to implement any of these functions, the specific

codes, T/R bit assignments, and use of a data word, shall be used as indicated. The use of the broadcast command option shall only be applied to particular mode codes as specified in table I.

4.3.3.5.1.7.1 Dynamic bus control. The controller shall issue a transmit command to an RT capable of performing the bus control function. This RT shall respond with a status word as specified in 4.3.3.5.3. Control of the data bus passes from the offering bus controller to the accepting RT upon completion of the transmission of the status word by the RT. If the RT rejects control of the data bus, the offering bus controller retains control of the data bus.

4.3.3.5.1.7.2 Synchronize (without data word). This command shall cause the RT to synchronize (e.g., to reset the internal timer, to start a sequence, etc.). The RT shall transmit the status word as specified in 4.3.3.5.3.

4.3.3.5.1.7.3 Transmit status word. This command shall cause the RT to transmit the status word associated with the last valid command word preceding this command. This mode command shall not alter the state of the status word.

4.3.3.5.1.7.4 Initiate self test. This command shall be used to initiate self test within the RT. The RT shall transmit the status word as specified in 4.3.3.5.3.

4.3.3.5.1.7.5 Transmitter shutdown. This command (to only be used with dual redundant bus systems) shall cause the RT to disable the transmitter associated with the redundant bus. The RT shall not comply with a command to shut down a transmitter on the bus from which this command is received. In all cases, the RT shall respond with a status word as specified in 4.3.3.5.3 after this command.

4.3.3.5.1.7.6 Override transmitter shutdown. This command (to only be used with dual redundant bus system) shall cause the RT to enable a transmitter which was previously disabled. The RT shall not comply with a command to enable a transmitter on the bus from which this command is received. In all cases, the RT shall respond with a status word as specified in 4.3.3.5.3 after this command.

4.3.3.5.1.7.7 Inhibit terminal flag (T/F) bit. This command shall cause the RT to set the T/F bit in the status word specified in 4.3.3.5.3 to logic zero until otherwise commanded. The RT shall transmit the status word as specified in 4.3.3.5.3.

4.3.3.5.1.7.8 Override inhibit T/F bit. This command shall cause the RT to override the inhibit T/F bit specified in 4.3.3.5.1.7.7. The RT shall transmit the status word as specified in 4.3.3.5.3.

4.3.3.5.1.7.9 Reset remote terminal. This command shall be used to reset the RT to a power up initialized state. The RT shall first transmit its status word, and then reset.

4.3.3.5.1.7.10 Reserved mode codes (01001 to 01111). These mode codes are reserved for future use and shall not be used.

TABLE I. Assigned mode codes

<u>T/R Bit</u>	<u>Mode Code</u>	<u>Function</u>	<u>Associated Data Word</u>	<u>Broadcast Command Allowed</u>
1	00000	Dynamic Bus Control	No	No
1	00001	Synchronize	No	Yes
1	00010	Transmit Status Word	No	No
1	00011	Initiate Self Test	No	Yes
1	00100	Transmitter Shutdown	No	Yes
1	00101	Override Transmitter Shutdown	No	Yes
1	00110	Inhibit Terminal Flag Bit	No	Yes
1	00111	Override Inhibit Terminal Flag bit	No	Yes
1	01000	Reset Remote Terminal	No	Yes
1	01001	Reserved	No	TBD
	↓	↓	↓	↓
1	01111	Reserved	No	TBD
1	10000	Transmit Vector Word	Yes	No
0	10001	Synchronize	Yes	Yes
1	10010	Transmit Last Command	Yes	No
1	10011	Transmit BIT Word	Yes	No
0	10100	Selected Transmitter Shutdown	Yes	Yes
0	10101	Override Selected Transmitter Shutdown	Yes	Yes
1 or 0	10110	Reserved	Yes	TBD
	↓	↓	↓	↓
1 or 0	11111	Reserved	Yes	TBD

NOTE: To be determined (TBD)

4.3.3.5.1.7.11 Transmit vector word. This command shall cause the RT to transmit a status word as specified in 4.3.3.5.3 and a data word containing service request information.

4.3.3.5.1.7.12 Synchronize (with data word). The RT shall receive a command word followed by a data word as specified in 4.3.3.5.2. The data word shall contain synchronization information for the RT. After receiving the command and data word, the RT shall transmit the status word as specified in 4.3.3.5.3.

4.3.3.5.1.7.13 Transmit last command word. This command shall cause the RT to transmit its status word as specified in 4.3.3.5.3 followed by a single data word which contains bits 4-19 of the last command word, excluding a transmit last command word mode code received by the RT. This mode command shall not alter the state of the RT's status word.

4.3.3.5.1.7.14 Transmit built-in-test (BIT) word. This command shall cause the RT to transmit its status word as specified in 4.3.3.5.3 followed by a single data word containing the RT BIT data. This function is intended to supplement the available bits in the status word when the RT hardware is sufficiently complex to warrant its use. The data word, containing the RT BIT data, shall not be altered by the reception of a transmit last command or a transmit status word mode code. This function shall not be used to convey BIT data from the associated subsystem(s).

4.3.3.5.1.7.15 Selected transmitter shutdown. This command shall cause the RT to disable the transmitter associated with a specified redundant data bus. The command is designed for use with systems employing more than two redundant buses. The transmitter that is to be disabled shall be identified in the data word following the command word in the format as specified in 4.3.3.5.2. The RT shall not comply with a command to shut down a transmitter on the bus from which this command is received. In all cases, the RT shall respond with a status word as specified in 4.3.3.5.3.

4.3.3.5.1.7.16 Override selected transmitter shutdown. This command shall cause the RT to enable a transmitter which was previously disabled. The command is designed for use with systems employing more than two redundant buses. The transmitter that is to be enabled shall be identified in the data word following the command word in the format as specified in 4.3.3.5.2. The RT shall not comply with a command to enable a transmitter on the bus from which this command is received. In all cases, the RT shall respond with a status word as specified in 4.3.3.5.3.

4.3.3.5.1.7.17 Reserved mode codes (10110 to 11111). These mode codes are reserved for future use and shall not be used.

4.3.3.5.2 Data word. A data word shall be comprised of a sync waveform, data bits, and a parity bit (see figure 3).

4.3.3.5.2.1 Sync. The data sync waveform shall be an invalid Manchester waveform as shown on figure 5. The width shall be three bit times, with the waveform being negative for the first one and one-half bit times, and then positive for the following one and one-half bit times. Note that if the bits preceding and following the sync are logic ones, then the apparent width of the sync waveform will be increased to four bit times.

MIL-STD-1553B
21 September 1978

4.3.3.5.2.2 Data. The sixteen bits following the sync shall be utilized for data transmission as specified in 4.3.2.

4.3.3.5.2.3 Parity. The last bit shall be utilized for parity as specified in 4.3.3.5.1.6.

4.3.3.5.3 Status word. A status word shall be comprised of a sync waveform, RT address, message error bit, instrumentation bit, service request bit, three reserved bits, broadcast command received bit, busy bit, subsystem flag bit, dynamic bus control acceptance bit, terminal flag bit, and a parity bit. For optional broadcast operation, transmission of the status word shall be suppressed as specified in 4.3.3.6.7.

4.3.3.5.3.1 Sync. The status sync waveform shall be as specified in 4.3.3.5.1.1.

4.3.3.5.3.2 RT address. The next five bits following the sync shall contain the address of the RT which is transmitting the status word as defined in 4.3.3.5.1.2.

4.3.3.5.3.3 Message error bit. The status word bit at bit time nine (see figure 3) shall be utilized to indicate that one or more of the data words associated with the preceding receive command word from the bus controller has failed to pass the RT's validity tests as specified in 4.4.1.1. This bit shall also be set under the conditions specified in 4.4.1.2, 4.4.3.4 and 4.4.3.6. A logic one shall indicate the presence of a message error, and a logic zero shall show its absence. All RT's shall implement the message error bit.

4.3.3.5.3.4 Instrumentation bit. The status word at bit time ten (see figure 3) shall be reserved for the instrumentation bit and shall always be a logic zero. This bit is intended to be used in conjunction with a logic one in bit time ten of the command word to distinguish between a command word and a status word. The use of the instrumentation bit is optional.

4.3.3.5.3.5 Service request bit. The status word bit at bit time eleven (see figure 3) shall be reserved for the service request bit. The use of this bit is optional. This bit when used, shall indicate the need for the bus controller to take specific predefined actions relative to either the RT or associated subsystem. Multiple subsystems, interfaced to a single RT, which individually require a service request signal shall logically OR their individual signals into the single status word bit. In the event this logical OR is performed, then the designer must make provisions in a separate data word to identify the specific requesting subsystem. The service request bit is intended to be used only to trigger data transfer operations which take place on an exception rather than periodic basis. A logic one shall indicate the presence of a service request, and a logic zero its absence. If this function is not implemented, the bit shall be set to zero.

4.3.3.5.3.6 Reserved status bits. The status word bits at bit times twelve through fourteen are reserved for future use and shall not be used. These bits shall be set to a logic zero.

4.3.3.5.3.7 Broadcast command received bit. The status word at bit time fifteen shall be set to a logic one to indicate that the preceding valid command word was a broadcast command and a logic zero shall show it was not a broadcast command. If the broadcast command option is not used, this bit shall be set to a logic zero.

4.3.3.5.3.8 Busy bit. The status word bit at bit time sixteen (see figure 3) shall be reserved for the busy bit. The use of this bit is optional. This bit, when used, shall indicate that the RT or subsystem is unable to move data to or from the subsystem in compliance with the bus controller's command. A logic one shall indicate the presence of a busy condition, and a logic zero its absence. In the event the busy bit is set in response to a transmit command, then the RT shall transmit its status word only. If this function is not implemented, the bit shall be set to logic zero.

4.3.3.5.3.9 Subsystem flag bit. The status word bit at bit time seventeen (see figure 3) shall be reserved for the subsystem flag bit. The use of this bit is optional. This bit, when used, shall flag a subsystem fault condition, and alert the bus controller to potentially invalid data. Multiple subsystems, interfaced to a single RT, which individually require a subsystem flag bit signal shall logically OR their individual signals into the single status word bit. In the event this logical OR is performed, then the designer must make provisions in a separate data word to identify the specific reporting subsystem. A logic one shall indicate the presence of the flag, and a logic zero its absence. If not used, this bit shall be set to logic zero.

4.3.3.5.3.10 Dynamic bus control acceptance bit. The status word bit at bit time eighteen (see figure 3) shall be reserved for the acceptance of dynamic bus control. This bit shall be used if the RT implements the optional dynamic bus control function. This bit, when used, shall indicate acceptance or rejection of a dynamic bus control offer as specified in 4.3.3.5.1.7.1. A logic one shall indicate acceptance of control, and a logic zero shall indicate rejection of control. If this function is not used, this bit shall be set to logic zero.

4.3.3.5.3.11 Terminal flag bit. The status word bit at bit time nineteen (see figure 3) shall be reserved for the terminal flag function. The use of this bit is optional. This bit, when used, shall flag a RT fault condition. A logic one shall indicate the presence of the flag, and a logic zero, its absence. If not used, this bit shall be set to logic zero.

4.3.3.5.3.12 Parity bit. The least significant bit in the status word shall be utilized for parity as specified in 4.3.3.5.1.6.

4.3.3.5.4 Status word reset. The status word bit, with the exception of the address, shall be set to logic zero after a valid command word is received by the RT with the exception as specified in 4.3.3.5.1.7. If the conditions which caused bits in the status word to be set (e.g., terminal flag) continue after the bits are reset to logic zero, then the affected status word bit shall be again set, and then transmitted on the bus as required.

4.3.3.6 Message formats. The messages transmitted on the data bus shall be in accordance with the formats on figure 6 and figure 7. The maximum and minimum response times shall be as stated in 4.3.3.7 and 4.3.3.8. No message formats, other than those defined herein, shall be used on the bus.

21 September 1978

4.3.3.6.1 Bus controller to remote terminal transfers. The bus controller shall issue a receive command followed by the specified number of data words. The RT shall, after message validation, transmit a status word back to the controller. The command and data words shall be transmitted in a contiguous fashion with no interword gaps.

4.3.3.6.2 Remote terminal to bus controller transfers. The bus controller shall issue a transmit command to the RT. The RT shall, after command word validation, transmit a status word back to the bus controller, followed by the specified number of data words. The status and data words shall be transmitted in a contiguous fashion with no interword gaps.

4.3.3.6.3 Remote terminal to remote terminal transfers. The bus controller shall issue a receive command to RT A followed contiguously by a transmit command to RT B. RT B shall, after command validation, transmit a status word followed by the specified number of data words. The status and data words shall be transmitted in a contiguous fashion with no gap. At the conclusion of the data transmission by RT B, RT A shall transmit a status word within the specified time period.

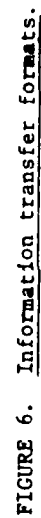
4.3.3.6.4 Mode command without data word. The bus controller shall issue a transmit command to the RT using a mode code specified in table I. The RT shall, after command word validation, transmit a status word.

4.3.3.6.5 Mode command with data word (transmit). The bus controller shall issue a transmit command to the RT using a mode code specified in table I. The RT shall, after command word validation, transmit a status word followed by one data word. The status word and data word shall be transmitted in a contiguous fashion with no gap.

4.3.3.6.6 Mode command with data word (receive). The bus controller shall issue a receive command to the RT using a mode code specified in table I, followed by one data word. The command word and data word shall be transmitted in a contiguous fashion with no gap. The RT shall, after command and data word validation, transmit a status word back to the controller.

4.3.3.6.7 Optional broadcast command. See 10.6 for additional information on the use of the broadcast command.

4.3.3.6.7.1 Bus controller to remote terminal(s) transfer (broadcast). The bus controller shall issue a receive command word with 1111 in the RT address field followed by the specified number of data words. The command word and data words shall be transmitted in a contiguous fashion with no gap. The RT(s) with the broadcast option shall after message validation, set the broadcast command received bit in the status word as specified in 4.3.3.5.3.7 and shall not transmit the status word.



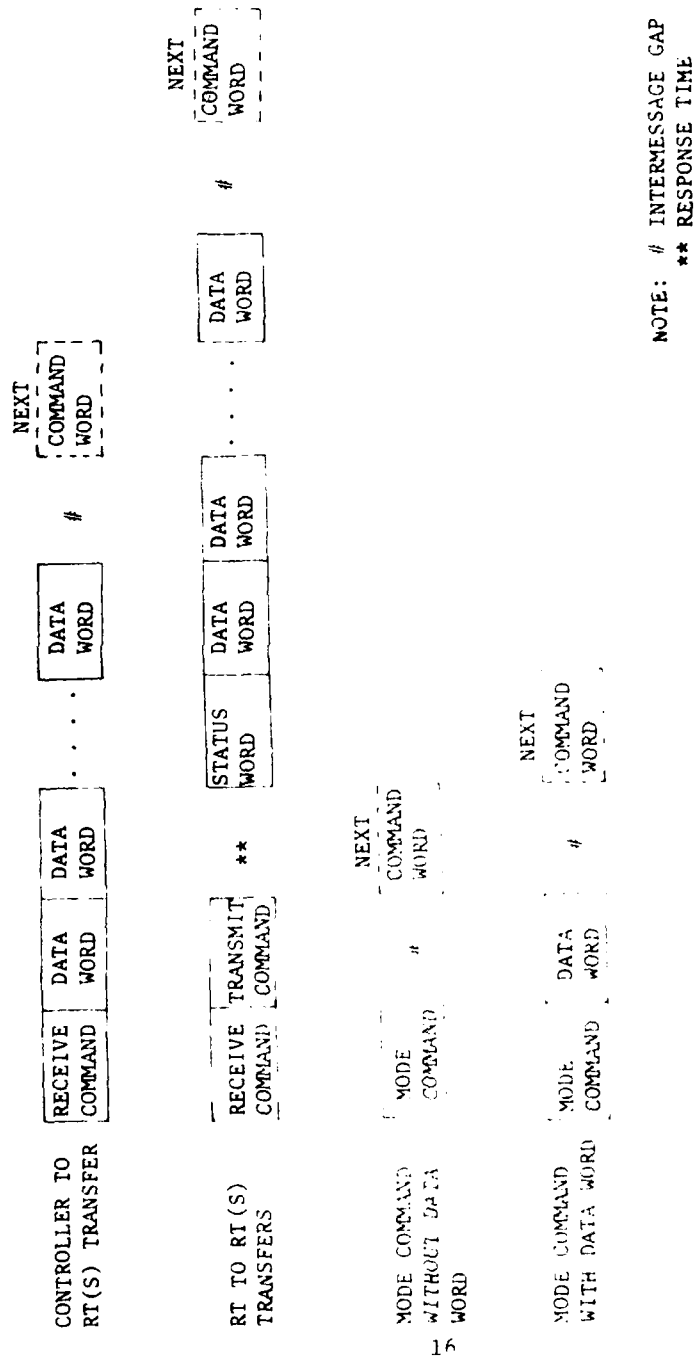


FIGURE 7. Broadcast information transfer formats.

4.3.3.6.7.2 Remote terminal to remote terminal(s) transfers (broadcast). The bus controller shall issue a receive command word with 11111 in the RT address field followed by a transmit command to RT A using the RT's address. RT A shall, after command word validation, transmit a status word followed by the specified number of data words. The status and data words shall be transmitted in a contiguous fashion with no gap. The RT(s) with the broadcast option, excluding RT A, shall after message validation, set the broadcast received bit in the status word as specified in 4.3.3.5.3.7 and shall not transmit the status word.

4.3.3.6.7.3 Mode command without data word (broadcast). The bus controller shall issue a transmit command word with 11111 in the RT address field, and a mode code specified in table I. The RT(s) with the broadcast option shall after command word validation, set the broadcast received bit in the status word as specified in 4.3.3.5.3.7 and shall not transmit the status word.

4.3.3.6.7.4 Mode command with data word (broadcast). The bus controller shall issue a receive command word with 11111 in the RT address field and a mode code specified in table I, followed by one data word. The command word and data word shall be transmitted in a contiguous fashion with no gap. The RT(s) with the broadcast option shall after message validation, set the broadcast received bit in the status word as specified in 4.3.3.5.3.7 and shall not transmit the status word.

4.3.3.7 Intermessage gap. The bus controller shall provide a minimum gap time of 4.0 microseconds (μ s) between messages as shown on figure 6 and figure 7. This time period, shown as T on figure 8, is measured at point A of the bus controller as shown on figure 9 or figure 10. The time is measured from the mid-bit zero crossing of the last bit of the preceding message to mid-zero crossing of the next command word sync.

4.3.3.8 Response time. The RT shall respond, in accordance with 4.3.3.6, to a valid command word within the time period of 4.0 to 12.0 μ s. This time period, shown as T on figure 8, is measured at point A of the RT as shown on figure 9 or figure 10. The time is measured from the mid bit zero crossing of the last word as specified in 4.3.3.6 and as shown on figure 6 and figure 7 to the mid-zero crossing of the status word sync.

4.3.3.9 Minimum no-response time-out. The minimum time that a terminal shall wait before considering that a response as specified in 4.3.3.8 has not occurred shall be 14.0 μ s. The time is measured from the mid-bit zero crossing of the last bit of the last word to the mid-zero crossing of the expected status word sync at point A of the terminal as shown on figure 9 or figure 10.

4.4 Terminal operation.

4.4.1 Common operation. Terminals shall have common operating capabilities as specified in the following paragraphs.

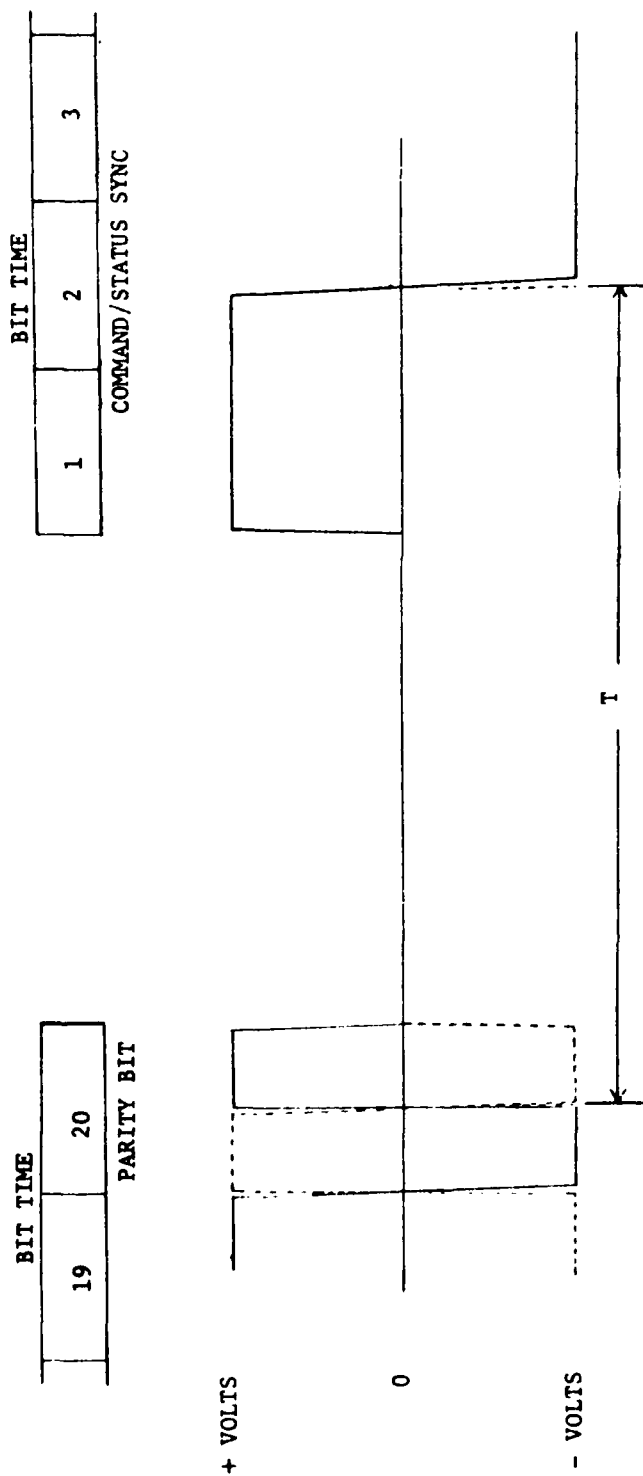


FIGURE 8. Intermessage gap and response time.

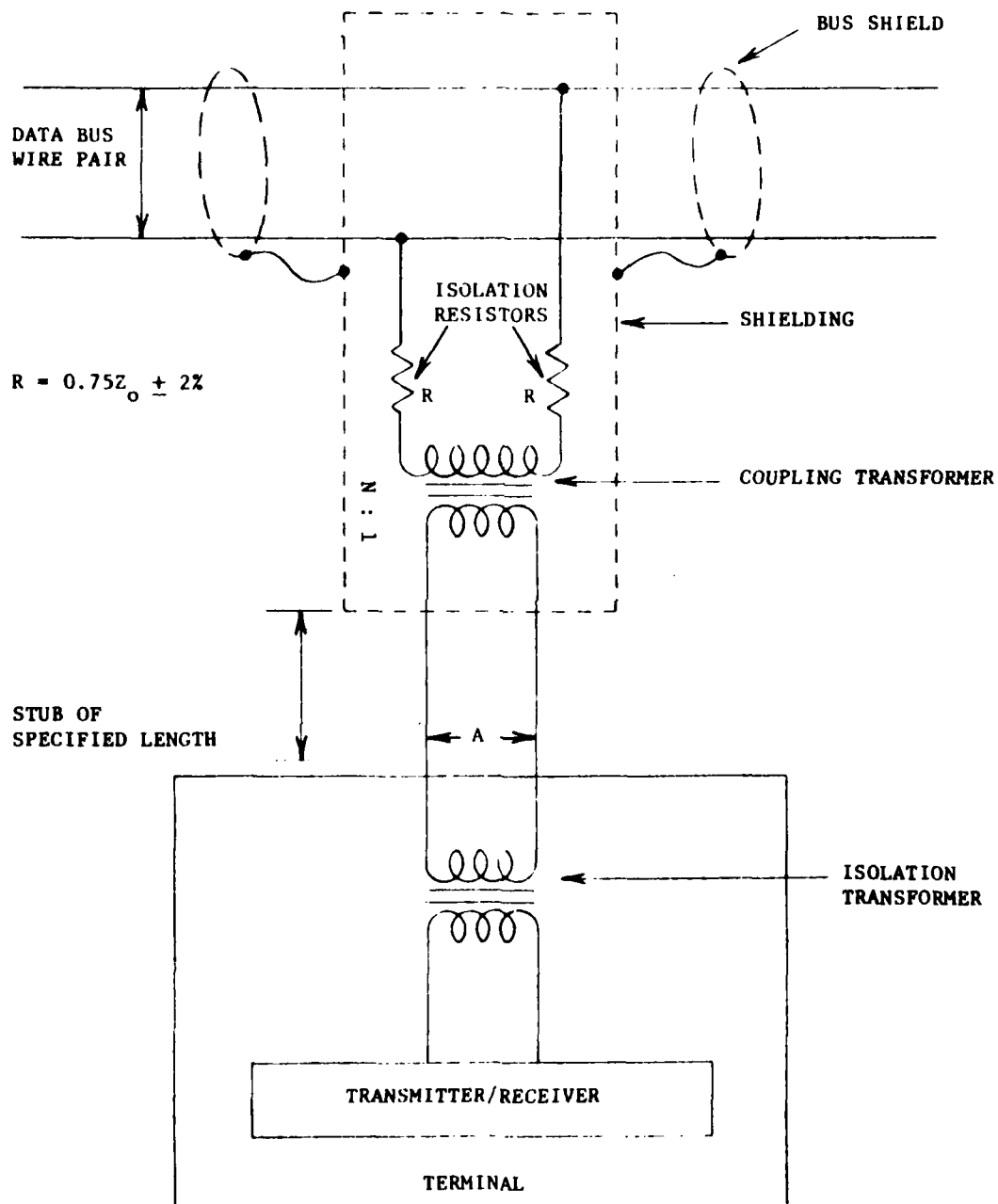


FIGURE 9. Data bus interface using transformer coupling.

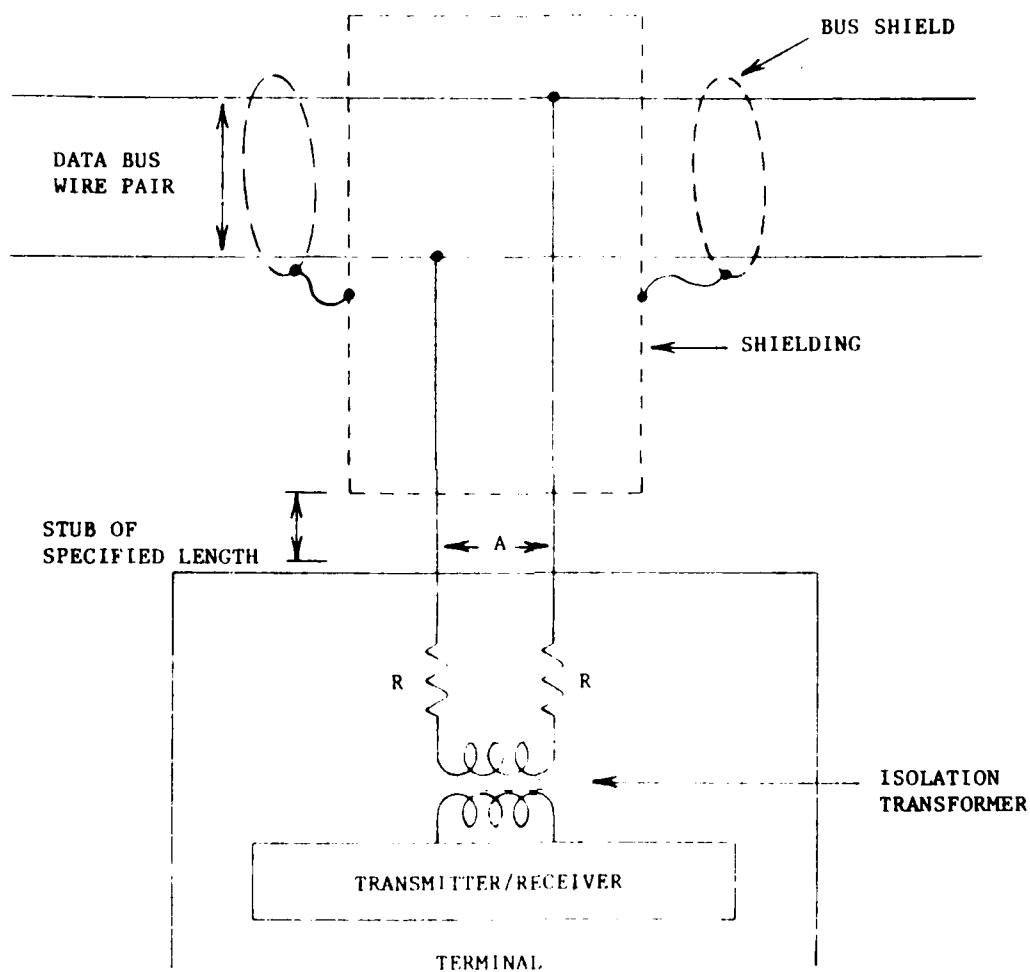


FIGURE 10. Data bus interface using direct coupling.

4.4.1.1 Word validation. The terminal shall insure that each word conforms to the following minimum criteria:

- a. The word begins with a valid sync field.
- b. The bits are in a valid Manchester II code.
- c. The information field has 16 bits plus parity.
- d. The word parity is odd.

When a word fails to conform to the preceding criteria, the word shall be considered invalid.

4.4.1.2 Transmission continuity. The terminal shall verify that the message is contiguous as defined in 4.3.3.6. Improperly timed data syncs shall be considered a message error.

4.4.1.3 Terminal fail-safe. The terminal shall contain a hardware implemented time-out to preclude a signal transmission of greater than 800.0 μ s. This hardware shall not preclude a correct transmission in response to a command. Reset of this time-out function shall be performed by the reception of a valid command on the bus on which the time-out has occurred.

4.4.2 Bus controller operation. A terminal operating as a bus controller shall be responsible for sending data bus commands, participating in data transfers, receiving status responses, and monitoring system status as defined in this standard. the bus controller function may be embodied as either a stand-alone terminal, whose sole function is to control the data bus(s), or contained within a subsystem. Only one terminal shall be in active control of a data bus at any one time.

4.4.3 Remote terminal.

4.4.3.1 Operation. A remote terminal (RT) shall operate in response to valid commands received from the bus controller. The RT shall accept a command word as valid when the command word meets the criteria of 4.4.1.1, and the command word contains a terminal address which matches the RT address or an address of 11111, if the RT has the broadcast option.

4.4.3.2 Superseding valid commands. The RT shall be capable of receiving a command word on the data bus after the minimum intermessage gap time as specified in 4.3.3.7 has been exceeded, when the RT is not in the time period T as specified in 4.3.3.8 prior to the transmission of a status word, and when it is not transmitting on that data bus. A second valid command word sent to an RT shall take precedence over the previous command. The RT shall respond to the second valid command as specified in 4.3.3.8.

4.4.3.3 Invalid commands. A remote terminal shall not respond to a command word which fails to meet the criteria specified in 4.4.3.1.

4.4.3.4 Illegal command. An illegal command is a valid command as specified in 4.4.3.1, where the bits in the subaddress/mode field, data word count/mode code field, and the T/R bit indicate a mode command, subaddress, or word count that has not been implemented in the RT. It is the responsibility of the bus controller to assure that no illegal commands are sent out. The RT designer has the option of monitoring for illegal commands. If an RT that is designed with this option detects an illegal command and the proper number of contiguous

21 September 1978

valid data words as specified by the illegal command word, it shall respond with a status word only, setting the message error bit, and not use the information received.

4.4.3.5 Valid data reception. The remote terminal shall respond with a status word when a valid command word and the proper number of contiguous valid data words are received, or a single valid word associated with a mode code is received. Each data word shall meet the criteria specified in 4.4.1.1.

4.4.3.6 Invalid data reception. Any data word(s) associated with a valid receive command that does not meet the criteria specified in 4.4.1.1 and 4.4.1.2 or an error in the data word count shall cause the remote terminal to set the message error bit in the status word to a logic one and suppress the transmission of the status word. If a message error has occurred, then the entire message shall be considered invalid.

4.4.4 Bus monitor operation. A terminal operating as a bus monitor shall receive bus traffic and extract selected information. While operating as a bus monitor, the terminal shall not respond to any message except one containing its own unique address if one is assigned. All information obtained while acting as a bus monitor shall be strictly used for off-line applications (e.g., flight test recording, maintenance recording or mission analysis) or to provide the back-up bus controller sufficient information to take over as the bus controller.

4.5 Hardware characteristics.

4.5.1 Data bus characteristics.

4.5.1.1 Cable. The cable used for the main bus and all stubs shall be a two conductor, twisted, shielded, jacketed cable. The wire-to-wire distributed capacitance shall not exceed 30.0 picofarads per foot. The cables shall be formed with not less than four twists per foot where a twist is defined as a 360 degree rotation of the wire pairs; and, the cable shield shall provide a minimum of 75.0 percent coverage.

4.5.1.2 Characteristic impedance. The nominal characteristic impedance of the cable (Z_0) shall be within the range of 70.0 ohms to 85.0 ohms at a sinusoidal frequency of 1.0 megahertz (MHz).

4.5.1.3 Cable attenuation. At the frequency of 4.5.1.2, the cable power loss shall not exceed 1.5 decibels (dB)/100 feet (ft).

4.5.1.4 Cable termination. The two ends of the cable shall be terminated with a resistance, equal to the selected cable nominal characteristic impedance (Z_0) $\pm$ 2.0 percent.

4.5.1.5 Cable stub requirements. The cable shall be coupled to the terminal as shown on figure 9 or figure 10. The use of long stubs is discouraged, and the length of a stub should be minimized. However, if installation requirements dictate, stub lengths exceeding those lengths specified in 4.5.1.5.1 and 4.5.1.5.2 are permissible.

4.5.1.5.1 Transformer coupled stubs. The length of a transformer coupled stub should not exceed 20 feet. If a transformer coupled stub is used, then the following shall apply.

4.5.1.5.1.1 Coupling transformer. A coupling transformer, as shown on figure 9, shall be required. This transformer shall have a turns ratio of $1:1.41 \pm 3.0$ percent, with the higher turns on the isolation resistor side of the stub.

4.5.1.5.1.1.1 Transformer input impedance. The open circuit impedance as seen at point B on figure 11 shall be greater than 3000 ohms over the frequency range of 75.0 kilohertz (kHz) to 1.0 megahertz (MHz), when measured with a 1.0 V root-mean-square (RMS) sin wave.

4.5.1.5.1.1.2 Transformer waveform integrity. The droop of the transformer using the test configuration shown on figure 11 at point B, shall not exceed 20.0 percent. Overshoot and ringing as measured at point B shall be less than ± 1.0 V peak. For this test, R shall equal 360.0 ohms ± 5.0 percent and the input A of figure 11 shall be a 250.0 kHz square wave, 27.0 V peak-to-peak, with a rise and fall time no greater than 100 nanoseconds (ns).

4.5.1.5.1.1.3 Transformer common mode rejection. The coupling transformer shall have a common mode rejection ratio greater than 45.0 dB at 1.0 MHz.

4.5.1.5.1.2 Fault isolation. An isolation resistor shall be placed in series with each connection to the data bus cable. This resistor shall have a value of $0.75 Z_0$ ohms plus or minus 2.0 percent, where Z_0 is the selected cable nominal characteristic impedance. The impedance placed across the data bus cable shall be no less than $1.5 Z_0$ ohms for any failure of the coupling transformer, cable stub, or terminal transmitter/receiver.

4.5.1.5.1.3 Cable coupling. All coupling transformers and isolation resistors, as specified in 4.5.1.5.1.1 and 4.5.1.5.1.2, shall have continuous shielding which will provide a minimum of 75 percent coverage. The isolation resistors and coupling transformers shall be placed at minimum possible distance from the junction of the stub to the main bus.

4.5.1.5.1.4 Stub voltage requirements. Every data bus shall be designed such that all stubs at point A of figure 9 shall have a peak-to-peak amplitude, line-to-line within the range of 1.0 and 14.0 V for a transmission by any terminal on the data bus. This shall include the maximum reduction of data bus signal amplitude in the event that one of the terminals has a fault which causes it to reflect a fault impedance specified in 4.5.1.5.1.2 on the data bus. This shall also include the worse case output voltage of the terminals as specified in 4.5.2.1.1.1 and 4.5.2.2.1.1.

4.5.1.5.2 Direct coupled stubs. The length of a direct coupled stub should not exceed 1 foot. Refer to 10.5 for comments concerning direct coupled stubs. If a direct coupled stub is used, then the following shall apply.

4.5.1.5.2.1 Fault isolation. An isolation resistor shall be placed in series with each connection to the data bus cable. This resistor shall have a value of 55.0 ohms plus or minus 2.0 percent. The isolation resistors shall be placed within the RT as shown on figure 10.

MIL-STD-1553B
21 September 1978

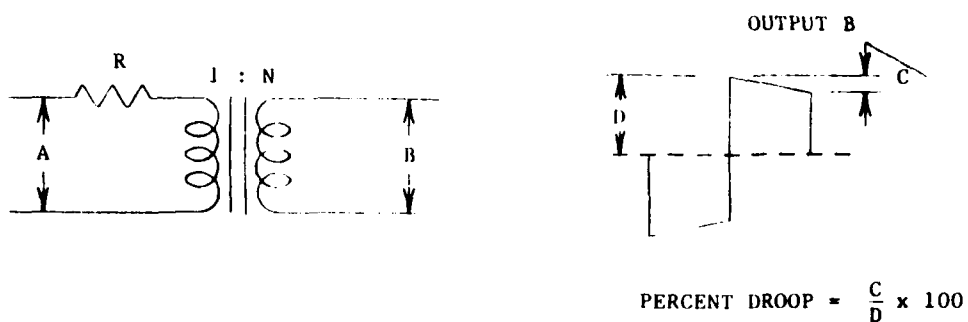


FIGURE 11. Coupling transformer.

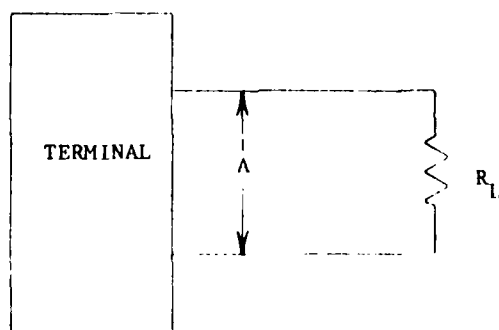


FIGURE 12. Terminal I/O characteristics for transformer coupled and direct coupled stubs.

4.5.1.5.2.2 Cable coupling. All bus-stub junctions shall have continuous shielding which will provide a minimum of 75 percent coverage.

4.5.1.5.2.3 Stub voltage requirements. Every data bus shall be designed such that all stubs at point A of figure 10 shall have a peak-to-peak amplitude, line-to-line within the range of 1.4 and 20.0 V for a transmission by any terminal on the data bus. This shall include the maximum reduction of data bus signal amplitude in the event that one of the terminals has a fault which causes it to reflect a fault impedance of 110 ohms on the data bus. This shall also include the worst case output voltage of the terminals as specified in 4.5.2.1.1.1 and 4.5.2.2.1.1.

4.5.1.5.3 Wiring and cabling for EMC. For purposes of electromagnetic capability (EMC), the wiring and cabling provisions of MIL-E-6051 shall apply.

4.5.2 Terminal characteristics.

4.5.2.1 Terminals with transformer coupled stubs.

4.5.2.1.1 Terminal output characteristics. The following characteristics shall be measured with R_L , as shown on figure 12, equal to 70.0 ohms $\pm$ 2.0 percent.

4.5.2.1.1.1 Output levels. The terminal output voltage levels shall be measured using the test configuration shown on figure 12. The terminal output voltage shall be within the range of 18.0 to 27.0 V, peak-to-peak, line-to-line, when measured at point A on figure 12.

4.5.2.1.1.2 Output waveform. The waveform, when measured at point A on figure 12 shall have zero crossing deviations which are equal to, or less than, 25.0 ns from the ideal crossing point, measured with respect to the previous zero crossing (i.e., $.5 \pm .025$ μ s, $1.0 \pm .025$ μ s, $1.5 \pm .025$ μ s, and $2.0 \pm .025$ μ s). The rise and fall time of this waveform shall be from 100.0 to 300.0 ns when measured from levels of 10 to 90 percent of full waveform peak-to-peak, line-to-line, voltage as shown on figure 13. Any distortion of the waveform including overshoot and ringing shall not exceed $\pm$ 900.0 millivolts (mV) peak, line-to-line, as measured at point A, figure 12.

4.5.2.1.1.3 Output noise. Any noise transmitted when the terminal is receiving or has power removed, shall not exceed a value of 14.0 mV, RMS, line-to-line, as measured at point A, figure 12.

4.5.2.1.1.4 Output symmetry. From the time beginning 2.5 μ s after the mid-bit crossing of the parity bit of the last word transmitted by a terminal, the maximum voltage at point A of figure 12 shall be no greater than $\pm$ 250.0 mV peak, line-to-line. This shall be tested with the terminal transmitting the maximum number of words it is designed to transmit, up to 33. This test shall be run six times with each word in a contiguous block of words having the same bit pattern. The six word contents that shall be used are 8000₁₆, 7FFF₁₆, 0000₁₆, FFFF₁₆, 5555₁₆, and AAAA₁₆. The output of the terminal shall be as specified in 4.5.2.1.1.1 and 4.5.2.1.1.2.

4.5.2.1.2 Terminal input characteristics. The following characteristics shall be measured independently.

MIL-STD-1553B
21 September 1978

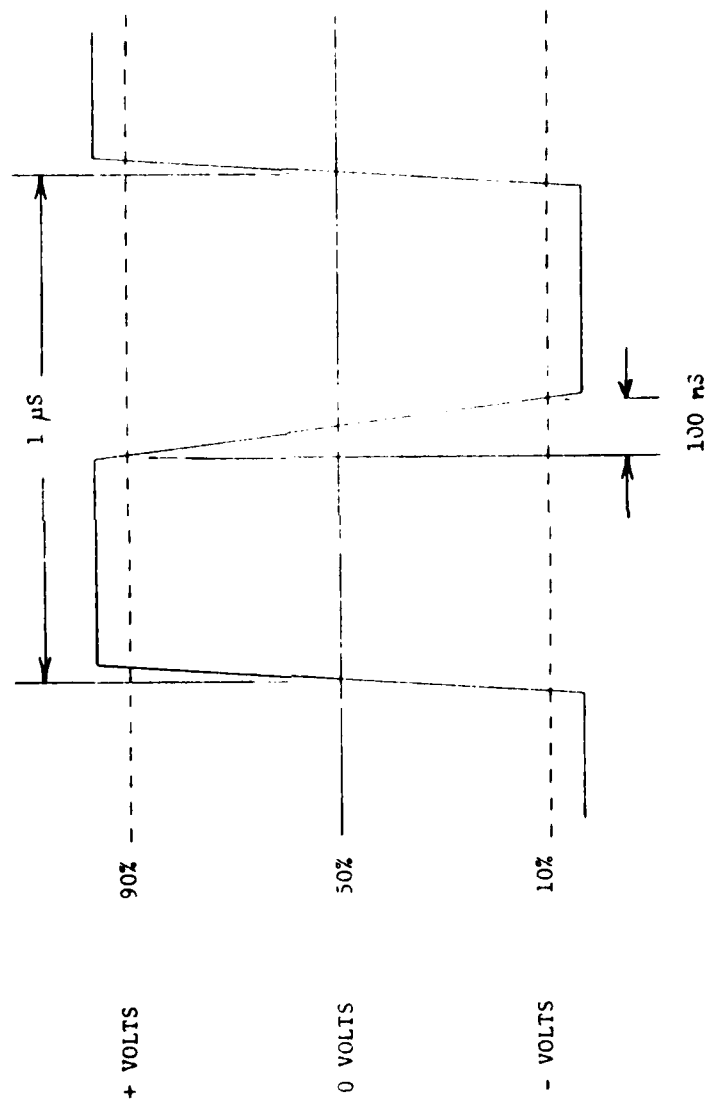


FIGURE 13. Output waveform.

4.5.2.1.2.1 Input waveform compatibility. The terminal shall be capable of receiving and operating with the incoming signals specified herein, and shall accept waveform varying from a square wave to a sine wave with a maximum zero crossing deviation from the ideal with respect to the previous zero crossing of ± 150 ns, (i.e., $2.0 \pm .15$ μ s, $1.5 \pm .15$ μ s, $1.0 \pm .15$ μ s, $.5 \pm .15$ μ s). The terminal shall respond to an input signal whose peak-to-peak amplitude, line-to-line, is within the range of .86 to 14.0 V. The terminal shall not respond to an input signal whose peak-to-peak amplitude, line-to-line, is within the range of 0.0 to .20 V. The voltages are measured at point A on figure 9.

4.5.2.1.2.2 Common mode rejections. Any signals from direct current (DC) to 2.0 MHz, with amplitudes equal to or less than ± 10.0 V peak, line-to-ground, measured at point A on figure 9, shall not degrade the performance of the receiver.

4.5.2.1.2.3 Input impedance. The magnitude of the terminal input impedance, when the RT is not transmitting, or has power removed, shall be a minimum of 1000.0 ohms within the frequency range of 75.0 kHz to 1.0 MHz. This impedance is that measured line-to-line at point A on figure 9.

4.5.2.1.2.4 Noise rejection. The terminal shall exhibit a maximum word error rate of one part in 10^7 , on all words received by the terminal, after validation checks as specified in 4.4, when operating in the presence of additive white Gaussian noise distributed over a bandwidth of 1.0 kHz to 4.0 MHz at an RMS amplitude of 140 mV. A word error shall include any fault which causes the message error bit to be set in the terminal's status word, or one which causes a terminal to not respond to a valid command. The word error rate shall be measured with a 2.1 V peak-to-peak, line-to-line, input to the terminal as measured at point A on figure 9. The noise tests shall be run continuously until, for a particular number of failures, the number of words received by the terminal, including both command and data words, exceeds the required number for acceptance of the terminal, or is less than the required number for rejection of the terminal, as specified in table II. All data words used in the tests shall contain random bit patterns. These bit patterns shall be unique for each data word in a message, and shall change randomly from message to message.

4.5.2.2 Terminals with direct coupled stubs.

4.5.2.2.1 Terminal output characteristics. The following characteristics shall be measured with R_L , as shown on figure 12, equal to 35.0 ohms ± 2.0 percent.

4.5.2.2.1.1 Output levels. The terminal output voltage levels shall be measured using the test configuration shown on figure 12. The terminal output voltage shall be within the range of 6.0 to 9.0 V, peak-to-peak, line-to-line, when measured at point A on figure 12.

MIL-STD-1553B
21 September 1978

Table II. Criteria for acceptance or rejection of a
terminal for the noise rejection test

TOTAL WORDS RECEIVED BY THE TERMINAL
(in multiples of 10^7)

No. of Errors	Reject (Equal or less)	Accept (Equal or more)
0	N/A	4.40
1	N/A	5.21
2	N/A	6.02
3	N/A	6.83
4	N/A	7.64
5	N/A	8.45
6	.45	9.27
7	1.26	10.08
8	2.07	10.89
9	2.88	11.70
10	3.69	12.51
11	4.50	13.32
12	5.31	14.13
13	6.12	14.94
14	6.93	15.75
15	7.74	16.56
16	8.55	17.37
17	9.37	18.19
18	10.18	19.00
19	10.99	19.81
20	11.80	20.62
21	12.61	21.43
22	13.42	22.24
23	14.23	23.05
24	15.04	23.86
25	15.85	24.67
26	16.66	25.48
27	17.47	26.29
28	18.29	27.11
29	19.10	27.92
30	19.90	28.73
31	20.72	29.54
32	21.53	30.35
33	22.34	31.16
34	23.15	31.97
35	23.96	32.78
36	24.77	33.00
37	25.58	33.00
38	26.39	33.00
39	27.21	33.00
40	28.02	33.00
41	33.00	N/A

4.5.2.2.1.2 Output waveform. The waveform, when measured at point A on figure 12, shall have zero crossing deviations which are equal to, or less than, 25.0 ns from the ideal crossing point, measured with respect to the previous zero crossing (i.e., $.5 \pm .025 \mu\text{s}$, $1.0 \pm .025 \mu\text{s}$, $1.5 \pm .025 \mu\text{s}$ and $2.0 \pm .025 \mu\text{s}$). The rise and fall time of this waveform shall be from 100.0 to 300.0 ns when measured from levels of 10 to 90 percent of full waveform peak-to-peak, line-to-line, voltage as shown on figure 13. Any distortion of the waveform including overshoot and ringing shall not exceed $\pm 300.0 \text{ mV}$ peak, line-to-line, as measured at point A on figure 12.

4.5.2.2.1.3 Output noise. Any noise transmitted when the terminal is receiving or has power removed, shall not exceed a value of 5.0 mV, RMS, line-to-line, as measured at point A on figure 12.

4.5.2.2.1.4 Output symmetry. From the time beginning 2.5 μs after the mid-bit crossing of the parity bit of the last word transmitted by a terminal, the maximum voltage at point A on figure 12, shall be no greater than $\pm 90.0 \text{ mV}$ peak, line-to-line. This shall be tested with the terminal transmitting the maximum number of words it is designed to transmit, up to 33. This test shall be run six times with each word in a contiguous block of words having the same bit pattern. The six word contents that shall be used are 8000₁₆, 7FFF₁₆, 0000₁₆, FFFF₁₆, 5555₁₆, and AAAA₁₆. The output of the terminal shall be as specified in 4.5.2.2.1.1 and 4.5.2.2.1.2.

4.5.2.2.2 Terminal input characteristics. The following characteristics shall be measured independently.

4.5.2.2.2.1 Input waveform compatibility. The terminal shall be capable of receiving and operating with the incoming signals specified herein, and shall accept waveform varying from a square wave to a sine wave with a maximum zero crossing deviation from the ideal with respect to the previous zero crossing of plus or minus 150 ns, (i.e., $2.0 \pm .15 \mu\text{s}$, $1.5 \pm .15 \mu\text{s}$, $1.0 \pm .15 \mu\text{s}$, $.5 \pm .15 \mu\text{s}$). The terminal shall respond to an input signal whose peak-to-peak amplitude, line-to-line, is within the range of 1.2 to 20.0 V. The terminal shall not respond to an input signal whose peak-to-peak amplitude, line-to-line, is within the range of 0.0 to .28 V. The voltages are measured at point A on figure 10.

4.5.2.2.2.2 Common mode rejections. Any signals from DC to 2.0 MHz, with amplitudes equal to or less than $\pm 10.0 \text{ V}$ peak, line-to-ground, measured at point A on figure 10, shall not degrade the performance of the receiver.

4.5.2.2.2.3 Input impedance. The magnitude of the terminal input impedance, when the RT is not transmitting, or has power removed, shall be a minimum of 2000.0 ohms within the frequency range of 75.0 kHz to 1.0 MHz. This impedance is that measured line-to-line at point A on figure 10.

4.5.2.2.2.4 Noise rejection. The terminal shall exhibit a maximum word error rate of one part in 10^7 , on all words received by the terminal, after validation checks as specified in 4.4, when operating in the presence of additive white Gaussian noise distributed over a bandwidth of 1.0 kHz to 4.0 MHz at an RMS amplitude of 200 mV. A word error shall include any fault which causes the message error bit to be set in the terminal's status word, or one which causes a terminal to not respond to a valid command. The word error rate shall be measured with a 3.0 V peak-to-peak, line-to-line, input to the

MIL-STD-1553B
21 September 1978

terminal as measured at point A on figure 10. The noise tests shall be run continuously until, for a particular number of failures, the number of words received by the terminal, including both command and data words, exceeds the required number for acceptance of the terminal, or is less than the required number for rejection of the terminal, as specified in table II. All data words used in the tests shall contain random bit patterns. These bit patterns shall be unique for each data word in a message, and shall change randomly from message to message.

4.6 Redundant data bus requirements. If redundant data buses are used, the requirements as specified in the following shall apply to those data buses.

4.6.1 Electrical isolation. All terminals shall have a minimum of 45 dB isolation between data buses. Isolation here means the ratio in dB between the output voltage on the active data bus and the output voltage on the inactive data bus. This shall be measured using the test configuration specified in 4.5.2.1.1 or 4.5.2.2.1 for each data bus. Each data bus shall be alternately activated with all measurements being taken at point A on figure 12 for each data bus.

4.6.2 Single event failures. All data buses shall be routed to minimize the possibility that a single event failure to a data bus shall cause the loss of more than that particular data bus.

4.6.3 Dual standby redundant data bus. If a dual redundant data bus is used, then it shall be a dual standby redundant data bus as specified in the following paragraphs.

4.6.3.1 Data bus activity. Only one data bus can be active at any given time except as specified in 4.6.3.2.

4.6.3.2 Reset data bus transmitter. If while operating on a command, a terminal receives another valid command, from either data bus, it shall reset and respond to the new command on the data bus on which the new command is received. The terminal shall respond to the new command as specified in 4.3.3.8.

5. DETAIL REQUIREMENTS (Not Applicable)

Custodians:
Army - EL
Navy - AS
Air Force - 11

Preparing Activity:
Air Force - 11
Project MISC-OD03

APPENDIX

10. General. The following paragraphs in this appendix are presented in order to discuss certain aspects of the standard in a general sense. They are intended to provide a user of the standard more insight into the aspects discussed.

10.1 Redundancy. It is intended that this standard be used to support rather than to supplant the system design process. However, it has been found, through application experience in various aircraft, that the use of a dual standby redundancy technique is very desirable for use in integrating mission avionics. For this reason, this redundancy scheme is defined in 4.6 of this standard. None the less, the system designer should utilize this standard as the needs of a particular application dictate. The use of redundancy, the degree to which it is implemented, and the form which it takes must be determined on an individual application basis. Figures 10.1 and 10.2 illustrate some possible approaches to dual redundancy. These illustrations are not intended to be inclusive, but rather representative. It should be noted that analogous approaches exist for the triple and quad redundant cases.

10.2 Bus controller. The bus controller is a key part of the data bus system. The functions of the bus controller, in addition to the issuance of commands, must include the constant monitoring of the data bus and the traffic on the bus. It is envisioned that most of the routine minute details of bus monitoring (e.g., parity checking, terminal non-response time-out, etc.) will be embodied in hardware, while the algorithms for bus control and decision making will reside in software. It is also envisioned that, in general, the bus controller will be a general purpose airborne computer with a special input/output (I/O) to interface with the data bus. It is of extreme importance in bus controller design that the bus controller be readily able to accommodate terminals of differing protocol's and status word bits used. Equipment designed to MIL-STD-1553A will be in use for a considerable period of time; thus, bus controllers must be capable of adjusting to their differing needs. It is also important to remember that the bus controller will be the focal point for modification and growth within the multiplex system, and thus the software must be written in such a manner as to permit modification with relative ease.

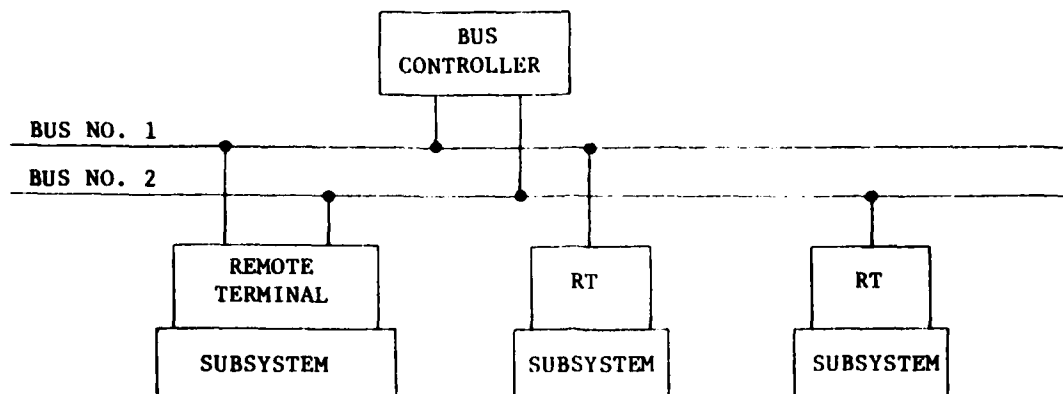


FIGURE 10.1. Illustration of possible redundancy.

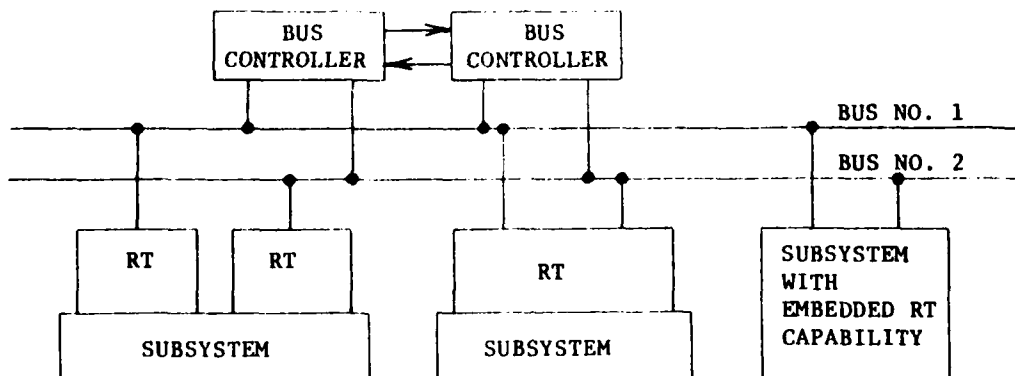


Figure 10.2

NOTE: RT - Remote Terminal

FIGURE 10.2. Illustration of possible redundancy.

10.3 Multiplex selection criteria. The selection of candidate signals for multiplexing is a function of the particular application involved, and criteria will in general vary from system to system. Obviously, those signals which have bandwidths of 400 Hz or less are prime candidates for inclusion on the bus. It is also obvious that video, audio, and high speed parallel digital signals should be excluded. The area of questionable application is usually between 400 Hz and 3KHz bandwidth. The transfer of these signals on the data bus will depend heavily upon the loading of the bus in a particular application. The decision must be based on projected future bus needs as well as the current loading. Another class of signals which in general are not suitable for multiplexing are those which can be typified by a low rate (over a mission) but possessing a high priority or urgency. Examples of such signals might be a nuclear event detector output or a missile launch alarm from a warning receiver. Such signals are usually better left hardwired, but they may be accommodated by the multiplex system if a direct connection to the bus controller's interrupt hardware is used to trigger a software action in response to the signal.

10.4 High reliability requirements. The use of simple parity for error detection within the multiplex bus system was dictated by a compromise between the need for reliable data transmission, system overhead, and remote terminal simplicity. Theoretical and empirical evidence indicates that an undetected bit error rate of 10^{-12} can be expected from a practical multiplex system built to this standard. If a particular signal requires a bit error rate which is better than that provided by the parity checking, then it is incumbent upon the system designer to provide the reliability within the constraints of the standard or to not include this signal within the multiplex bus system. A possible approach in this case would be to have the signal source and sink provide appropriate error detection and correction encoding/decoding and employ extra data words to transfer the information. Another approach would be to partition the message, transmit a portion at a time, and then verify (by interrogation) the proper transfer of each segment.

10.5 Stubbing. Stubbing is the method wherein a separate line is connected between the primary data bus line and a terminal. The direct connection of a stub line causes a mismatch which appears on the waveforms. This mismatch can be reduced by filtering at the receiver and by using bi-phase modulation. Stubs are often employed not only as a convenience in bus layout but as a means of coupling a unit to the line in such a manner that a fault on the stub or terminal will not greatly affect the transmission line operation. In this case, a network is employed in the stub line to provide isolation from the fault. These networks are also used for stubs that are of such length that the mismatch and reflection degrades bus operation. The preferred method of stubbing is to use transformer coupled stubs, as defined in 4.5.1.5.1. This method provides the benefits of DC isolation, increased common mode protection, a doubling of effective stub impedance, and fault isolation for the entire stub and terminal. Direct coupled stubs, as defined in 4.5.1.5.2 of this standard, should be avoided if at all possible. Direct coupled stubs provide no DC isolation or common mode rejection for the terminal external to its subsystem. Further, any shorting fault between the subsystems internal isolation resistors (usually on a circuit board) and the main bus junction will cause failure of that entire bus. It can be expected that when the direct coupled stub length exceeds 1.6 feet, that it will begin to distort the main bus waveforms. Note that this length includes the cable runs internal to a given subsystem.

MIL-STD-1553b
21 September 1978

10.6 Use of broadcast option. The use of a broadcast message as defined in 4.3.3.6.7 of this standard represents a significant departure from the basic philosophy of this standard in that it is a message format which does not provide positive closed-loop control of bus traffic. The system designer is strongly encouraged to solve any design problems through the use of the three basic message formats without resorting to use of the broadcast. If system designers do choose to use the broadcast command, they should carefully consider the potential effects of a missed broadcast message, and the subsequent implications for fault or error recovery design in the remote terminals and bus controllers.

FOLD

ASD/ENESS
Wright-Patterson AFB, O 45433
OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE \$300

POSTAGE AND FEES PAID
DEPARTMENT OF THE AIR FORCE
DoD-318



ASD/ENESS
Wright-Patterson AFB, Ohio 45433

FOLD

| This document has been approved |
| for public release and sale; its |
| distribution is unlimited |

MIL-STD-1589B (USAF)
06 June 1980
SUPERSEDING
MIL-STD-1589A (USAF)
15 March 1979

MILITARY STANDARD

JOVIAL (J73)

FSC IPSC

MIL-STD-1589B(USAF)
06 June 1980

PREFACE

This document is the revised MIL-STD-1589B Draft (USAF) definition of the upgraded J73 JOVIAL programming language. The sections are organized in a top-down manner. The first section describes the interactions between the modules of the complete program so that in subsequent sections the structures of the language can be described (to the extent possible) without reference to their interaction with other structures.

Most sections are divided into separate parts entitled "Syntax," "Semantics," and "Constraints." The "Syntax" descriptions define the grammar of the language in a modified BNF notation. The "Semantics" discussions define the meaning of constructs that satisfy the Syntax and Constraints. The "Constraints" discussions enumerate non-syntactic requirements that must be met in order for the given constructs to be legal. The intent is that the Syntax, Semantics, and Constraints not be redundant with each other - e.g., the Semantics sections do not normally repeat something that should be obvious from the Syntax, neither do they repeat stipulations that are listed as Constraints.

Some of the designated Constraints apply at compile time, and others pertain to errors that are not detectable until the compiled program is executed. In order to conform to this standard, a J73 compiler must detect compile-time errors, but it is not required to generate code for run-time checks.

The Appendix provides a cross-reference index to constructs that appear in the Syntax. For each construct, the index gives the number of the section where that construct is defined and the numbers of the sections where that construct is used in a definition.

The following metalanguage conventions have been observed in this document:

1. Terminal symbols, i.e., those which actually appear in a program are written in upper case. For example:

BEGIN
END
STATIC

2. Non-terminal symbols, i.e., those which represent groups of terminal symbols are written in lower case and enclosed between < and >. If any non-terminal symbol is longer than one word, the words are separated by a hyphen. For example:

<compool-module>
<ordinary-table-body>

3. The following special symbols are used in the metalanguage.

`::=` means "is defined as." For example,

`<a> ::= <b> <c>`

where `<a>` is defined as the string `<b>` followed by the string `<c>`. Definitions that do not fit on one line may extend to the next line or lines.

| The | symbol indicates that what follows is an alternate choice of definition for the non-terminal to the left of the `::=` symbol. For example,

`<a> ::= <b> | <c>`

where `<a>` is defined as either the string `<b>` or the string `<c>`.

[] If a string may optionally be present, it is enclosed between [and]. For example,

`<a> ::= [<b>] <c>`

where `<a>` is defined as either the string `<c>` or the string `<b>` followed by the string `<c>`.

4. The following symbols have metalinguistic meaning when appended to a non-terminal:

... One or more instances of the string represented by non-terminal

,... One or more instances separated by a comma

:... One or more instances separated by a colon

For example:

`<a>...` Represents a single `<a>` or any sequence of `<a>`'s (e.g., `<a>` or `<a> <a>` or `<a> <a> <a>` etc.)

`[<a>...]` Represents the null string or any sequence of `<a>`'s

`<a>,...` Represents a single `<a>` or any length sequence of `<a>`'s separated by commas (e.g., `<a>` or `<a>,<a>` or

<a>,<a>,<a> etc.)

5. If a non-terminal appearing on the right side of the ::= is not defined in that same sub-section, the number of the sub-section where it is defined appears in parentheses in the right margin.
6. In a "Semantics" or "Constraints" section, non-terminal symbols are enclosed between < and > when the usage refers to constructs occurring in a "syntax" section or when the specific J73 meaning might be confused with generalized programming usage.

Throughout this document, the symbols used for the prime, the quotation mark, and a blank are as follows:

1. Prime
2. Quotation mark "
3. Blank space

MIL-STD-1589B (USAF)
06 June 1980

TABLE OF CONTENTS

	<u>Page</u>
1.0 <u>Global Concepts</u>	1
1.1 The Complete Program	1
1.2 Modules	1
1.2.1 Compool Modules	1
1.2.2 Procedure Modules	2
1.2.3 Main Program Modules	3
1.2.4 Conditional Compilation	4
1.3 Scope of Names	4
1.4 Implementation Parameters	5
2.0 <u>Declarations</u>	12
2.1 Data Declarations	13
2.1.1 Item Declarations	14
2.1.1.1 Integer Type Descriptions	15
2.1.1.2 Floating Type Descriptions	16
2.1.1.3 Fixed Type Descriptions	18
2.1.1.4 Bit Type Descriptions	20
2.1.1.5 Character Type Descriptions	21
2.1.1.6 Status Type Descriptions	21
2.1.1.7 Pointer Type Descriptions	23
2.1.2 Table Declarations	24
2.1.2.1 Table Dimension Lists	25
2.1.2.2 Table Structure	27
2.1.2.3 Ordinary Table Entries	28
2.1.2.4 Specified Table Entries	30
2.1.3 Constant Declarations	33
2.1.4 Block Declarations	34
2.1.5 Allocation of Data Objects	35
2.1.6 Initialization of Data Objects	36
2.2 Type Declarations	38
2.3 Statement Name Declarations	41
2.4 Define Declarations	41

MIL-STD-1589B (USAF)
06 June 1980

2.4.1	Define Calls	43
2.5	External Declarations	45
2.5.1	DEF Specifications	45
2.5.2	REF Specifications	47
2.6	Overlay Declarations	48
2.7	Null Declarations	50
3.0	<u>Procedures and Functions</u>	51
3.1	Procedures	52
3.2	Functions	53
3.3	Parameters of Procedures and Functions	55
3.4	Inline Procedures and Functions	58
3.5	Machine-Specific Procedures and Functions	59
4.0	<u>Statements</u>	61
4.1	Assignment Statements	62
4.2	Loop Statements	63
4.3	IF Statements	66
4.4	CASE Statements	67
4.5	Procedure Call Statements	69
4.6	RETURN Statements	71
4.7	GOTO Statements	71
4.8	EXIT Statements	72
4.9	STOP Statements	72
4.10	ABORT Statements	73
5.0	<u>Formulas</u>	74
5.1	Numeric Formulas	76
5.1.1	Integer Formulas	76
5.1.2	Floating Formulas	78
5.1.3	Fixed Formulas	80
5.2	Bit Formulas	83
5.2.1	Relational Expressions	85
5.2.2	Boolean Formulas	87
5.3	Character Formulas	87
5.4	Status Formulas	88
5.5	Pointer Formulas	88

5.6	Table Formulas	90
6.0	<u>Data References</u>	91
6.1	Variables	91
6.2	Named Constants	96
6.3	Function Calls	97
6.3.1	LOC Function	98
6.3.2	NEXT Function	99
6.3.3	BIT Function	100
6.3.4	BYTE Function	101
6.3.5	Shift Functions	101
6.3.6	ABS Function	102
6.3.7	Sign Function	102
6.3.8	Size Functions	103
6.3.9	Bounds Functions	104
6.3.10	NWDSEN Function	105
6.3.11	Status Inverse Functions	105
7.0	<u>Type Matching and Conversions</u>	107
8.0	<u>Basic Elements</u>	116
8.1	Characters	116
8.2	Symbols	118
8.2.1	Names	118
8.2.2	Reserved Words	119
8.2.3	Operators	120
8.2.4	Separators	122
8.3	Literals	123
8.3.1	Numeric Literals	123
8.3.2	Bit Literals	125
8.3.3	Boolean Literals	128
8.3.4	Character Literals	128
8.3.5	Pointer Literals	128
8.4	Comments	129
8.5	Blanks	129
9.0	<u>Directives</u>	130

MIL-STD-1589B (USAF)
06 June 1980

9.1	Compool Directives	131
9.2	Text Directives	132
	9.2.1 Copy Directives	132
	9.2.2 Skip, Begin, and End Directives	133
9.3	Linkage Directives	133
9.4	Trace Directives	134
9.5	Interference Directives	135
9.6	Reducible Directives	136
9.7	Listing Directives	137
	9.7.1 Source-listing Directives	137
	9.7.2 Define-listing Directives	137
9.8	Register Directives	138
9.9	Expression Evaluation Order Directives	139
9.10	Initialization Directives	139
9.11	Allocation Order Directives	140
	APPENDIX - CROSS REFERENCE INDEX	142

1.0 GLOBAL CONCEPTS

1.1 THE COMPLETE PROGRAM

Syntax:

<complete-program>	::=	<module>...	
<module>	::=	<compool-module>	(1.2.1)
		<procedure-module>	(1.2.2)
		<main-program-module>	(1.2.3)

Semantics:

A <complete-program> of the J73 language gives the complete specification of a computational algorithm to be performed. A <complete-program> consists of a group of one or more <modules> that are compilable separately and which may be subsequently bound together for execution as a unit. A <module> is the smallest entity in the language that may be separately compiled.

A <complete-program> may contain zero or more <compool-modules> and zero or more <procedure-modules>.

Constraint:

A <complete-program> must contain exactly one <main-program-module>.

Note:

A compiler may accept a file containing more than one <module>, but it is not required to do so. If it does accept such a file, it must process each <module> as though it had been submitted separately.

1.2 MODULES

1.2.1 COMPOOL MODULES

Syntax:

<compool-module>	::=	START [<directive>...]	(9.0)
		COMPOOL <compool-name> ;	
		[<compool-declaration>...]	(2.0)
		[<directive>...]	(9.0)
		TERM	

<compool-name> ::= <name> (8.2.1)

Semantics:

<Compool-modules> provide a means of declaring data objects, types, and subroutines that are to be made external - i.e., that are potentially available to other <modules> in the <complete-program>. Another <module> may access the names declared in a given <compool-module> by use of a <compool-directive> (see Section 9.1) that names the given compool or by use of external declarations (see Section 2.5).

A <compool-module> may contain <compool-directives> that name other <compool-modules>.

By appropriate use of <def-specifications> and <ref-specifications> within <compool-declarations>, a user can control whether physical allocation takes place within the <compool-module> itself or within the accessing <module> (see Section 2.5).

1.2.2 PROCEDURE MODULES

Syntax:

```
<procedure-module> ::= START
                        {<declaration>...}          (2.0)
                        [<non-nested-subroutine>...]
                        [<directive>...]            (9.0)
                        TERM

<non-nested-subroutine> ::= [<directive>...]        (9.0)
                           [DEF] <subroutine-definition> (3.0)
```

Semantics:

<Procedure-modules> provide a means of separately compiling subroutines that specify portions of the actions of the <complete-program>.

If a <subroutine-definition> is preceded by DEF, that subroutine may be invoked from within the <main-program-module> or from within another <procedure-module>, provided that the referencing module contains an appropriate <ref-specification> for the subroutine or accesses a compool containing such a specification.

<Non-nested-subroutines> defined without a DEF may be invoked only from within the <procedure-module> or <main-program-module> in which

06 June 1980

they are defined. Similarly, all declarations in a `<procedure-module>` apply only within that `<procedure-module>` (unless they are `<external-declarations>` - see Section 2.5).

1.2.3 MAIN PROGRAM MODULES

Syntax:

<main-program-module>	::=	START [<directive>...]	(9.0)
		PROGRAM	
		<program-name> ;	
		[<directive>...]	(9.0)
		<program-body>	
		{<non-nested-subroutine>...}	(1.2.2)
		[<directive>...]	(9.0)
		TERM	
<program-name>	::=	<name>	(8.2.1)
<program-body>	::=	<statement>	(4.0)
		BEGIN [<declaration>...]	(2.0)
		<statement>...	(4.0)
		[<subroutine-definition>...]	(3.0)
		[<directive>...]	(9.0)
		[<label>...] END	(4.0)

Semantics:

The body of a <main-program-module> is executed at the start of a <complete-program>. When execution of the body is complete, execution of the <complete-program> is complete. Unless the <complete-program> consists of a single <main-program-module>, the <main-program-module> will contain one or more <compool-directives>, references to externally-declared data, and/or calls of DEF'd subroutines in other modules.

Declarations in a <main-program-module> may be external or internal. If a <non-nested-subroutine> has a DEF, it may be invoked either locally or from within a <procedure-module>, provided that the referencing module contains an appropriate <ref-specification> for the subroutine or accesses a compool containing such a specification. If it does not have a DEF, it can be invoked only from within the module in which it is defined.

Constraints:

The <program-body> must contain at least one non-null statement (e.g., STOP).

1.2.4 CONDITIONAL COMPILATION

Two methods are provided for conditionally suppressing generation of object code for portions of a JOVIAL module.

The !SKIP, !BEGIN, and !END directives (see Section 9.2.2) permit almost complete suppression of processing of suppressed source. The only processing done for suppressed source is to scan for the terminating !END directive. Therefore the suppressed source may contain errors and/or statements incompatible with other module source without affecting compilation.

The IF and CASE statements (see Sections 4.3 and 4.4) permit suppression of generation of object code. Source for this suppressed object code must be correct since it is subject to the same validity checks and processing of directives as other source code. Only code that is unconditionally unreachable is suppressed so this conditional compilation must produce the same results as if the code was generated. Segments of code which are unreachable due to values of <if-statement> <boolean-formulas> or <case-selector-formulas> which are <compile-time-formulas> and which do not contain <labels> are always suppressed. Implementations may choose to do a more complete analysis and also suppress other recognized unreachable code.

1.3 SCOPE OF NAMES

<Procedure-modules> and the <main-program-module> can contain subroutines (i.e., procedures and functions) nested to any depth. Each subroutine, as well as the <program-body> and the <main-program-module> or <procedure-module> itself, establishes a region or scope for which a name's declaration is active and in which the <name> can be used. The scope of a <name> is that region of the <complete-program> within which that <name> has a single meaning.

A name declared with a DEF or REF (see Section 2.5) is considered to be external; all other names are internal. An external <name> can be used in any module of the <complete-program>, except within a scope containing an internal name with the same spelling. An internal name can be used only within the subroutine, <procedure-module>, or <main-program-module> within which that name is declared, but not within an enclosed scope containing a <name> with the same spelling.

The <name> of a subroutine belongs to the scope in which that subroutine is declared or defined.

For any given compilation, all names made available from referenced <compool-modules> (see Section 9.1), as well as the name of the <module> being compiled and all <compool-names>, belong to the same scope, referred to as compool scope, which is considered to enclose the scope established by the <procedure-module>, <main-program-module>. or <compool-module> being compiled.

System-defined names (e.g., machine-specific subroutines, implementation parameters) belong to system scope, which encloses the compool scope. Such names may be redefined by the programmer.

These rules ensure that any two names with the same spelling but with distinct scopes are regarded as if they were different names.

Constraints:

No two names having the same scope may have the same spelling. (This constraint does not prevent two tables with different <table-names> to be declared in the same scope using the same <table-type-name>. See Sections 2.1.2 and 2.2.)

No two external names may have the same spelling.

1.4 IMPLEMENTATION PARAMETERS

Syntax:

```
<integer-machine-  
parameter> ::= BITSINBYTE  
              | BITSINWORD  
              | LOCSINWORD  
              | BYTEPOS  
                ( <compile-time-integer-formula> ) (5.1.1)  
              | BYTESINWORD  
              | BITSINPOINTER  
              | INTPRECISION  
              | FLOATPRECISION
```

	FIXEDPRECISION	
	FLOATRADIX	
	IMPLFLOATPRECISION (<precision>)	(2.1.1.2)
	IMPLFIXEDPRECISION (<scale-specifier> , <fraction-specifier>)	(2.1.1.3) (2.1.1.3)
	IMPLINTSIZE (<integer-size>)	(2.1.1.1)
	MAXFLOATPRECISION	
	MAXFIXEDPRECISION	
	MAXINTSIZE	
	MAXBYTES	
	MAXBITS	
	MAXINT (<integer-size>)	(2.1.1.1)
	MININT (<integer-size>)	(2.1.1.1)
	MAXTABLESIZE	
	MAXSTOP	
	MINSTOP	
	MAXSIGDIGITS	
	MINSIZE (<compile-time-integer-formula>)	(5.1.1)
	MINFRACTION (<compile-time-floating-formula>)	(5.1.2)
	MINSCALE (<compile-time-floating-formula>)	(5.1.2)
	MINRELPRECISION (<compile-time-floating-formula>)	(5.1.2)

```
<floating-machine-  
parameter> ::= MAXFLOAT ( <precision> )      (2.1.1.2)  
              | MINFLOAT ( <precision> )      (2.1.1.2)  
              | FLOATRELPRECISION  
                ( <precision> )                (2.1.1.2)  
              | FLOATUNDERFLOW  
                ( <precision> )                (2.1.1.2)  
  
<fixed-machine-  
parameter> ::= MAXFIXED ( <scale-specifier> , (2.1.1.3)  
                        <fraction-specifier> ) (2.1.1.3)  
              | MINFIXED ( <scale-specifier> , (2.1.1.3)  
                        <fraction-specifier> ) (2.1.1.3)
```

Semantics:

The machine on which a J73 program runs contains an array of memory cells. These cells are grouped or partitioned into the following units for purposes of the language specification.

1. Bit - The smallest unit of storage (can contain one of two values, which are represented by zero and one)
2. Byte - A group of one or more consecutive bits that is capable of holding a single character of information
3. Word - A memory partition of one or more consecutive bits that serves as the unit of allocation of data storage
4. Address Unit - The machine dependent unit used to identify an address or location in memory

The number of bits per byte, word, and address varies from implementation to implementation, and these quantities affect the representation and behavior of data in the language. Machine parameters are constants that describe these implementation-dependent differences. The values of these constants must be specified as part of the implementation of a J73 compiler on any computer. These names can then be referenced by a user to access the values associated with that implementation.

The size of an <integer-machine-parameter> is the size of an <integer-literal> having that value. The attributes of a <floating-machine-parameter> or <fixed-machine-parameter> are as specified by its <precision> or its <scale-specifier> and <fraction-specifier>. The

values of the implementation parameters are as follows:

BITSINBYTE	Number of bits in a byte
BITSINWORD	Number of bits in a word
LOCSINWORD	Number of locations (address units) in a word
BYTEPOS(PP)	A permitted <starting-bit> value for character strings that cross word boundaries. PP is any integer value between 0 and BYTESINWORD-1, inclusive and BYTEPOS(PP) < BYTEPOS(PP+1).
BYTESINWORD	Number of complete bytes in a word
BITSINPOINTER	Number of bits used for a pointer value
INTPRECISION	The number of bits that an implementation supplies to hold the value of an integer item (exclusive of sign, if any) when no <integer-size> is specified by the programmer.
FLOATPRECISION	The number of bits that an implementation supplies to hold the value of the mantissa of a floating point item (exclusive of the sign bit) when no <precision> is specified by the programmer
FIXEDPRECISION	The number of bits that an implementation supplies to hold the value of a fixed item (exclusive of the sign bit) when no <fraction-specifier> is supplied by the programmer
FLOATRADIX	Base of the floating point representation, specified as an integer
IMPLFLOATPRECISION(II)	Number of bits (not including the sign bit) in the mantissa of the representation for a floating point

	value whose specified precision is II
IMPLFIXEDPRECISION(SS,FF)	The number of bits (excluding sign bit) an implementation uses to represent an unpacked fixed item with scale SS and fraction FF. This value also determines the accuracy of fixed formula results.
IMPLINTSIZE(II)	The number of bits (excluding sign bit) an implementation uses to represent an unpacked S or U item with specified size II.
MAXFLOATPRECISION	Maximum specifiable precision supported by an implementation for a <floating-item-description>
MAXFIXEDPRECISION	Maximum value supported by an implementation for the sum of the scale and fraction specifiers in a <fixed-item-description>
MAXINTSIZE	Maximum specifiable size (not including the sign bit) supported by an implementation for signed and unsigned integers
MAXBYTES	Maximum value supported by an implementation for a <character-size>; must not exceed MAXBITS/BITSINBYTE
MAXBITS	Maximum value supported by an implementation for a <bit-size>; the maximum value of words per entry in a table is MAXBITS/BITSINWORD, and the maximum BITSIZE of a table entry is MAXBITS
MAXINT(SS)	Maximum integer value representable in SS+1 bits (including sign bit)
MININT(SS)	Minimum signed integer value representable in SS+1 bits (including sign bit), using the implementation's method of representing negative numbers

MAXTABLESIZE	The maximum number of words an implementation permits a table to occupy.
MAXSTOP	Maximum specifiable value for an <integer-formula> in a <stop-statement> (see Section 4.9)
MINSTOP	Minimum specifiable value for an <integer-formula> in a <stop-statement> (see Section 4.9)
MAXSIGDIGITS	The maximum number of significant digits an implementation will process for a fixed or floating point literal (see Section 8.3.1)
MINSIZE(II)	The minimum value of SS such that II is less than or equal to MAXINT(SS) and greater than or equal to MININT(SS)
MINFRACTION(AA)	The minimum value of FF such that $2^{*(-FF)}$ is less than or equal to AA. The value of AA must be greater than zero.
MINSCALE(AA)	The minimum value of SS such that 2^{*SS} is greater than AA. The value of AA must be greater than zero.
MINRELPRECISION(FF)	The minimum value of PP such that FLOATRELPRECISION(PP) is less than or equal to FF. The value of FF must be greater than or equal to FLOATRELPRECISION (MAXFLOATPRECISION).
MAXFLOAT(PP)	Maximum floating point value using only the first PP mantissa bits (excluding sign) of the implementation's floating point representation whose actual mantissa length is IMPLFLOATPRECISION(PP). PP must be greater than zero and not exceed MAXFLOATPRECISION.
MINFLOAT(PP)	Minimum floating point value representable in exactly PP mantissa

bits, (excluding sign) and using the implementation's method of representing negative numbers. PP must be greater than zero and not exceed MAXFLOATPRECISION.

FloatRelPrecision(PP)

Let FRP1 be the smallest floating point value greater than 1.0 using the first PP bits (excluding sign) of the implementation's representation for floating point values. FloatRelPrecision(PP) equals FRP1 - 1.0. PP must be greater than zero and not exceed MAXFLOATPRECISION.

FloatUnderflow(PP)

The smallest positive floating point value using exactly PP mantissa bits (excluding sign) and such that both FloatUnderflow(PP) and -FloatUnderflow(PP) are representable as floating point values

MaxFixed(SS,FF)

Maximum fixed value representable in SS+FF+1 bits (including sign bit)

MinFixed(SS,FF)

Minimum fixed value representable in SS+FF+1 bits (including sign bit), using the implementation's method for representing negative values

Note:

A FIXEDRADIX implementation parameter is not provided since fixed point values are represented using radix 2 (see Section 2.1.1.3).

2.0 DECLARATIONS

Syntax:

<declaration>	::=	<data-declaration>	(2.1)
		<type-declaration>	(2.2)
		<subroutine-declaration>	(3.0)
		<statement-name-declaration>	(2.3)
		<define-declaration>	(2.4)
		<external-declaration>	(2.5)
		<overlay-declaration>	(2.6)
		<inline-declaration>	(3.4)
		<null-declaration>	(2.7)
		BEGIN <declaration>... END	
		<directive> <declaration>	(9.0)
<compool-declaration>	::=	<external-declaration>	(2.5)
		<constant-declaration>	(2.1.3)
		<type-declaration>	(2.2)
		<define-declaration>	(2.4)
		<overlay-declaration>	(2.6)
		<null-declaration>	(2.7)
		BEGIN <compool-declaration>... END	
		<directive> <compool-declaration>	(9.0)

Semantics:

<Declarations> associate <names> with programmer-supplied meanings.

A `<compool-declaration>` is a `<declaration>` that appears in a `<compool-module>`.

Constraints

Except for `<statement-names>`, names of subroutines, type names in `<pointer-item-descriptions>`, and formal parameter names, a name may not be used prior to the point at which a `<declaration>` for that name appears.

2.1 DATA DECLARATIONS

Syntax:

<code><data-declaration></code>	<code>::= <item-declaration></code>	(2.1.1)
	<code><table-declaration></code>	(2.1.2)
	<code><constant-declaration></code>	(2.1.3)
	<code><block-declaration></code>	(2.1.4)

Semantics:

`<Data-declarations>` declare `<data-names>` and their attributes. Three kinds of data structures exist in J73:

1. Item - A simple data object of the language. An item is a variable of a pre-defined or programmer-defined type having no constituents.
2. Table - An aggregate data object consisting of a collection of one or more items, or an array of such collections. The collection of items is called an entry. An entire entry in a table is selected by the use of the table name, together with a sequence of indices ("subscripts") if the table is arrayed. An item within an entry is selected by the use of the item name and the appropriate number of subscripts.
3. Block - A group of items and tables and other blocks to which is allocated a contiguous area of storage.

Additionally, an item or table may be declared to be `CONSTANT`, in which case its value cannot be changed during execution. A constant item must be given an initial value by means of an `<item-preset>`. Blocks, items, or tables (other than constants) can specify, by means of an `<allocation-specifier>`, the allocation permanence of the storage associated with their names. Non-constant items and tables can

optionally be given initial values by means of <item-presets> or <table-presets>.

The value of an uninitialized data object is undefined until it receives a value in an executable statement.

Declarations associate a <name> with a type. A type determines the set of values that an object can have and the operations that can be performed on those values. Types are grouped into related sets called type classes. Examples of type classes are signed integer, unsigned integer, float, and bit. Types within a type class are distinguished by the values of certain properties known as attributes. For example, S 3 is a particular type within type class S with a value of 3 for the integer size attribute. Rules concerning type matching are found in Section 7.0.

2.1.1 ITEM DECLARATIONS

Syntax:

<item-declaration>	::=	ITEM <item-name> [<allocation-specifier>] <item-type-description> [<item-preset>] ;	(2.1.5) (2.1.6)
<item-name>	::=	<name>	(8.2.1)
<item-type-description>	::=	<integer-type-description> <floating-type-description> <fixed-type-description> <bit-type-description> <character-type-description> <status-type-description> <pointer-type-description>	(2.1.1.1) (2.1.1.2) (2.1.1.3) (2.1.1.4) (2.1.1.5) (2.1.1.6) (2.1.1.7)

Semantics:

<Item-declarations> declare items. Items are used as variables to retain values in a J73 program. Allocation for items declared in <item-declarations> will be such that no items share a word.

The <item-type-description> establishes the type of an item.

The <allocation-specifier> establishes the allocation permanence of items which are not enclosed in blocks. This allocation permanence is automatic if the declaration is in a subroutine and the <allocation-specifier> is omitted, otherwise it is STATIC (see Section 2.1.5). Items enclosed in blocks inherit the allocation permanence of the enclosing block.

The <item-preset>, if present, specifies an initial value for the item.

Constraints:

Only items having STATIC allocation (explicitly or by default) may contain an <item-preset>.

Declarations of items that are <formal-input-parameters> or <formal-output-parameters> (see Section 3.3) must not contain an <allocation-specifier> or <item-preset>.

An <item-declaration> within a block must not contain an <allocation-specifier>.

2.1.1.1 INTEGER TYPE DESCRIPTIONS

Syntax:

<integer-type-description>	::= <integer-item-description> <integer-type-name>	
<integer-item-description>	::= S [<round-or-truncate>] [<integer-size>]	(2.1.1.2)
	U [<round-or-truncate>] [<integer-size>]	(2.1.1.2)
<integer-size>	::= <comp-integer-formula>	(5.1.1)
<integer-type-name>	::= <item-type-name>	(2.2)

Semantics:

An <integer-type-description> is used to specify a signed integer type or an unsigned integer type. S specifies a signed integer type; U specifies an unsigned integer type.

The <integer-size> attribute specifies the minimum number of bits of storage required to hold the maximum value of the integer (excluding the sign, if any). If <integer-size> is omitted, it defaults to INTPRECISION. The number of bits allocated for signed integers will be at least <integer-size>+1, and for unsigned integers will be at least <integer-size>.

The value set for a signed integer type with size SS is MININT(SS) through MAXINT(SS). The value set for an unsigned integer type with size SS is 0 through MAXINT(SS).

The <round-or-truncate> attribute specifies truncation or rounding is to occur when a value is converted to an integer type. If R is specified, rounding will occur. If T is specified, truncation towards minus infinity will occur. If Z is specified, truncation towards zero will occur. If the attribute is omitted, truncation in an implementation-dependent manner will occur.

Constraints:

The maximum value that can be specified for <integer-size> is MAXINTSIZE, an implementation parameter.

<Integer-size> must be greater than zero.

An <integer-type-name> must be an <item-type-name> declared in an <item-type-declaration> that contains an <integer-type-description> (see Section 2.2).

Notes:

An implementation may choose MAXINTSIZE > BITSINWORD-1.

The <round-or-truncate> option has a use only when an <integer-item-description> is used in an <integer-conversion> (see Section 7.0).

2.1.1.2 FLOATING TYPE DESCRIPTIONS

Syntax:

```
<floating-type-description> ::= <floating-item-description>
                                | <floating-type-name>

<floating-item-description> ::= F [<round-or-truncate>]
                                [<precision>]
```

`<round-or-truncate>` ::= , R
 | , T
 | , Z

`<precision>` ::= `<compile-time-integer-formula>(5.1.1)`

`<floating-type-name>` ::= `<item-type-name>` (2.2)

Semantics:

A `<floating-type-description>` is used to specify a floating type. The `<precision>` attribute specifies the minimum number of bits of storage required to hold the value of the mantissa. If `<precision>` is omitted, it defaults to `FLOATPRECISION`, an implementation parameter.

The `<round-or-truncate>` attribute is used to specify whether truncation or rounding is to occur when a value of a floating type with a greater `<precision>` is assigned to an item of this type. If R is specified, rounding will occur. If T is specified, truncation towards minus infinity will occur. If Z is specified, truncation towards zero will occur. If the attribute is omitted, truncation in an implementation-dependent manner will occur. Rounding and truncation take place with respect to the implemented precision of the floating type. (Note: `IMPLFLOATPRECISION(PP)` is an implementation parameter defining what precision is provided when precision PP is specified.)

The value set for a floating type with `<precision>` PP is `MINFLOAT(PP)` through `-FLOATUNDERFLOW(PP)`, 0, and `FLOATUNDERFLOW(PP)` through `MAXFLOAT(PP)`.

Constraints:

The maximum value that can be specified for `<precision>` is `MAXFLOATPRECISION`, an implementation parameter.

`<Precision>` must be greater than zero.

A `<floating-type-name>` must be an `<item-type-name>` declared in an `<item-type-declaration>` that contains a `<floating-type-description>` (see Section 2.2).

Note:

Since a `<floating-type-description>` specifies only the minimum precision required, an implementation is free to support only one or two levels of implemented precision. Which implemented precision level represents a floating type depends on the value of the specified

precision. The implemented precision must never be less than the specified precision. Since an implementation may provide more than the specified precision, it is consistent to round or truncate a represented value only if converting from a longer to a shorter implemented precision.

2.1.1.3 FIXED TYPE DESCRIPTIONS

Syntax:

```
<fixed-type-description> ::= <fixed-item-description>
                             | <fixed-type-name>

<fixed-item-description> ::= A [<round-or-truncate>]      (2.1.1.2)
                             <scale-specifier>
                             [, <fraction-specifier>]

<scale-specifier>          ::= <compile-time-integer-formula>(5.1.1)

<fraction-specifier>       ::= <compile-time-integer-formula>(5.1.1)

<fixed-type-name>         ::= <item-type-name>            (2.2)
```

Semantics:

A <fixed-type-description> is used to specify a fixed point numeric type. If SS is the value of the <scale-specifier> and FF is the value of the <fraction-specifier>, then SS+FF is the minimum number of bits in the representation, excluding the sign bit. When SS and FF are both positive, SS specifies the number of bits to the left of the binary point (excluding the sign bit) and FF the minimum number of bits to the right (see Note below). When SS is negative, the binary point is assumed to be ABS(SS) bits to the left of the first (non-sign) bit of the representation. Similarly, when FF is negative, the least significant bit of the representation is no more than ABS(FF) bits to the left of the binary point.

The (nominal) precision of a fixed point type is the sum of its scale and fraction specifier. The implemented precision may be greater than the nominal bits required. If <fraction-specifier> is omitted, the fixed point type has a default precision given by FIXEDPRECISION, an implementation parameter, and the implied value of the omitted <fraction-specifier> is FIXEDPRECISION-SS, where SS is the <scale-specifier>.

If FF is a fixed point item declared with a default <fraction-specifier>, then $\text{FIXEDPRECISION} = \text{BITSIZE}(\text{REP}(\text{FF})) - 1$.

The <round-or-truncate> attribute specifies truncation or rounding is to occur when a value is converted to a fixed point type. If R is specified, rounding will occur. If T is specified, truncation towards minus infinity will occur. If Z is specified, truncation towards zero will occur. If the attribute is omitted, truncation in an implementation-dependent manner will occur. Rounding and truncation take place with respect to the implemented precision of the fixed type (see Note below).

The value set of a fixed point type with scale SS and fraction FF is MINFIXED(SS,FF) through MAXFIXED(SS,FF).

Constraints:

The sum of the scale and fraction specifiers (i.e., the nominal precision) must be greater than zero and must not exceed MAXFIXEDPRECISION, an implementation parameter.

The value of <scale-specifier> must lie in the range -127 through +127.

A <fixed-type-name> must be an <item-type-name> declared in an <item-type-declaration> that contains a <fixed-type-description> (see Section 2.2).

Notes:

The set of exactly representable fixed point values is determined by a fixed type's scale and fraction specifiers. A <fraction-specifier> value, FF, means fixed point values must be represented with a precision greater than or equal to $2^{*(-FF)}$. A <scale-specifier> value, SS, means the maximum representable value is at least $2^{*SS} - 2^{*(-FF)}$ and less than 2^{*SS} .

An implementation is permitted to support more than one level of implemented precision for fixed point types. For computational purposes, values will be represented using the smallest implemented precision level (e.g., one word or two words) consistent with the value's nominal precision. For storage purposes in packed tables, a fixed point value need occupy no more than the number of bits specified by the nominal precision plus one bit for the sign.

IMPLFIXEDPRECISION(SS,FF) is an implementation parameter defining what precision is provided for an unpacked fixed point item when nominal precision SS+FF is specified. In addition, the implemented precision of a packed item (i.e., an item in a specified table, packed ordinary table, or a tight table) as well as an unpacked item is given by $\text{BITSIZE}(\text{REP}(\text{FI}))-1$, where FI is the fixed point item.

The implemented precision of a fixed item is the number of bits (excluding sign bit) used to store the item. Assignments to such items round or truncate with respect to this precision, which is never less than the specified precision. Rounding or truncation can change a fixed point value only if the implemented precision is shortened.

It should be noted that specifying R, T, or Z in an item declaration only affects the conversion of literal values (see Section 8.3.1) and assignments of fixed point values when the stored representation of the value is shorter than the representation used for computations.

2.1.1.4 BIT TYPE DESCRIPTIONS

Syntax:

```
<bit-type-description> ::= <bit-item-description>
                           | <bit-type-name>

<bit-item-description> ::= B [<bit-size>]

<bit-size>                ::= <compile-time-integer-formula> (5.1.1)

<bit-type-name>           ::= <item-type-name> (2.2)
```

Semantics:

A <bit-type-description> is used to specify a bit string type. The <bit-size> attribute specifies the number of bits in the string. If <bit-size> is omitted it defaults to 1.

Constraints:

The maximum value that can be specified for <bit-size> is MAXBITS, an implementation parameter. The minimum value that can be specified for <bit-size> is one.

A <bit-type-name> must be an <item-type-name> declared in an <item-type-declaration> that contains a <bit-type-description> (see Section 2.2).

2.1.1.5 CHARACTER TYPE DESCRIPTIONS

Syntax:

```
<character-type-description> ::= <character-item-description>
                                | <character-type-name>

<character-item-description> ::= C [<character-size>]

<character-size>                ::= <compile-time-integer-formula>(5.1.1)

<character-type-name>           ::= <item-type-name>                (2.2)
```

Semantics:

A <character-type-description> is used to specify a fixed-length character string type. The <character-size> attribute specifies the number of characters in the string. If <character-size> is omitted it defaults to 1.

Constraints:

The maximum value that can be specified for <character-size> is MAXBYTES, an implementation parameter. The minimum value that can be specified for <character-size> is one.

A <character-type-name> must be an <item-type-name> declared in an <item-type-declaration> that contains a <character-type-description> (see Section 2.2).

2.1.1.6 STATUS TYPE DESCRIPTIONS

Syntax:

```
<status-type-description> ::= <status-item-description>
                                | <status-type-name>

<status-item-description> ::= STATUS [<status-size>]
                                ( <status-list> )

<status-list>                ::= <default-sublist>
                                | [<default-sublist> ,]
                                  <specified-sublist>,...
```

<default-sublist>	::= <status-constant>,...	
<specified-sublist>	::= <status-list-index> <status-constant>,...	
<status-list-index>	::= <compile-time-integer-formula>	(5.1.1)
<status-constant>	::= V (<status>)	
<status>	::= <name>	(8.2.1)
	<letter>	(8.1)
	<reserved-word>	(8.2.2)
<status-type-name>	::= <item-type-name>	(2.2)
<status-size>	::= <compile-time-integer-formula>	(5.1.1)

Semantics:

A <status-type-description> is used to specify a status type. The <status-list> is used to define the value set of the type, which consists of a set of named <status-constants>. These named <status-constants> are considered to be the logical values of the status type. Associated with each logical value is a representational value, i.e., how the value is actually represented internally. If the <status-list> contains only a <default-sublist>, the status type is said to have a default representation. The <status-constants> in the <default-sublist> will be assigned representational values 0 through N-1 (where N is the number of <status-constants> in the sublist) in the order in which they are specified in the list. The <status-constants> in each <specified-sublist> will be assigned representational values <status-list-index> through <status-list-index> + N-1 (where N is the number of <status-constants> in the sublist) in the order in which they are specified.

For a given <status-list>, the value of any <status-constant> is considered to be greater than the value of another <status-constant> having a lower representational value.

<Status-size> specifies the minimum number of bits to be allocated to hold the status value (excluding the sign bit, if any). If it is omitted, it defaults to the minimum needed for the representation as an integer value. If the representation of the lowest-valued <status-constant> in the list is less than zero, signed integer representation will be used; otherwise, unsigned integer representation will be used.

Constraints:

The <status-constants> must be unique within the <status-list>.

The <status-list-indices> within a <status-list> must be specified such that all the <status-constants> in the <status-list> receive unique representational values.

The value specified in <status-size> must be greater than or equal to the minimum needed for the representation of the status values and less than or equal to MAXINTSIZE.

The representation of a status value cannot be less than MININT (BITSINWORD-1) and it cannot exceed MAXINT(BITSINWORD-1).

A <status-type-name> must be an <item-type-name> declared in an <item-type-declaration> that contains a <status-type-description> (see Section 2.2).

Note:

The use of a <name> in a <status> does not constitute a declaration of the <name> or a reference to a declared <name> having the same spelling. Within a given scope, a <status> <name> and a declared <name> can have the same spelling and no conflict will result.

2.1.1.7 POINTER TYPE DESCRIPTIONS

Syntax:

<pointer-type-description>	::=	<pointer-item-description>	
		<pointer-type-name>	
<pointer-item-description>	::=	P {<type-name>}	
<pointer-type-name>	::=	<item-type-name>	(2.2)
<type-name>	::=	<item-type-name>	(2.2)
		<table-type-name>	(2.2)
		<block-type-name>	(2.2)

Semantics:

A <pointer-type-description> is used to specify a pointer type. If the <pointer-item-description> contains a <type-name>, then the pointer

being specified is a typed pointer. If the <type-name> is omitted, then the pointer is an untyped pointer.

A typed pointer contains the address of a data object of the type specified by the <type-name>. The object being pointed to may be obtained by dereferencing the pointer (see Section 6.1).

An untyped pointer contains the address of a data object of any type. However, such a pointer must be converted to a typed pointer (see Section 7.0) before it may be dereferenced or assigned to a typed pointer.

Constraint:

A <pointer-type-name> must be an <item-type-name> declared in an <item-type-declaration> that contains a <pointer-type-description> (see Section 2.2).

2.1.2 TABLE DECLARATIONS

Syntax:

<table-declaration>	::=	TABLE <table-name> [<allocation-specifier>] [<dimension-list>] table-description	(2.1.5) (2.1.2.1)
<table-description>	::=	[<structure-specifier>] <entry-specifier> <table-type-name> [<table-preset>] ;	(2.1.2.2) (2.2) (2.1.6)
<entry-specifier>	::=	<ordinary-entry-specifier> <specified-entry-specifier>	(2.1.2.3) (2.1.2.4)
<table-name>	::=	<name>	(8.2.1)

Semantics:

<Table-declarations> declare named aggregate data objects. The presence of a <dimension-list> indicates that the table is an arrayed collection of entries. The <dimension-list> specifies the range of indices of the array.

The <allocation-specifier> establishes the allocation permanence of tables which are not enclosed in blocks. This allocation permanence is

automatic if the declaration is in a subroutine and the <allocation-specifier> is omitted, otherwise it is STATIC (see Section 2.1.5). Tables enclosed in blocks inherit the allocation permanence of the enclosing block.

The <table-description> describes the contents of the table either with a <table-type-name> (see Section 2.2) or with an <entry-specifier>. Two or more tables may be declared in the same scope using the same <table-type-name>, and no name conflicts of the contained items will result, provided the <table-names> are different. Items in tables declared with a <table-type-name> can only be accessed using pointers to the tables (see Section 6.1).

A table may either be an ordinary table, in which only the logical structure is described (see Section 2.1.2.3) or a specified table, in which the detailed physical layout of the table is described (see Section 2.1.2.4).

A <structure-specifier> is used to specify the representation of entries in a dimensioned table (see Section 2.1.2.2).

The <table-preset>, if present, specifies initial values for the table components. For <table-descriptions> containing an <entry-specifier> rather than a <table-type-name>, the <table-preset> is part of the <entry-specifier> (see Section 2.1.2.3 and 2.1.2.4).

Constraints:

Only tables having STATIC allocation (explicitly or by default) may contain a <table-preset>.

Tables that are <formal-input-parameters> or <formal-output-parameters> (see Section 3.3) must not contain an <allocation-specifier> or <table-preset>.

A <table-declaration> within a block must not contain an <allocation-specifier>.

A dimensioned <table-declaration> must not contain a <table-type-name> whose declaration also contains a <dimension-list>.

A <structure-specifier> in an undimensioned table is prohibited.

2.1.2.1 TABLE DIMENSION LISTS

Syntax:

<dimension-list> ::= (<dimension>,...)

<dimension> ::= [<lower-bound-option>]
 <upper-bound>

 | *

<lower-bound-option> ::= <lower-bound> :

<lower-bound> ::= <compile-time-integer-formula> (5.1.1)
 | <compile-time-status-formula> (5.4)

<upper-bound> ::= <compile-time-integer-formula> (5.1.1)
 | <compile-time-status-formula> (5.4)

Semantics:

A <dimension-list> specifies that a table is an array. Each <dimension> specifies the range of values for that dimension. If the <lower-bound> is omitted, it defaults to zero if the <upper-bound> is an integer; if the <upper-bound> is a status value, it defaults to the first <status-constant> in the status type of the <upper-bound>.

A <dimension> of * that appears with a formal parameter means the bounds will be determined from the actual parameter on each invocation. (Note that in accordance with Sections 6.3.9 and 6.1, bounds of * dimensions range from 0 to NN-1, where NN is the number of elements in the corresponding dimension of the actual parameter, regardless of what the lower and upper bounds values are for the actual parameter or whether the bound has an integer or status type.)

Constraints:

Only status types with default representations may be used in <dimensions>.

The <lower-bound> must be less than or equal to the <upper-bound>.

The <lower-bound> and <upper-bound> must both be status formulas of the same type or both be integer formulas.

The maximum number of <dimensions> is seven.

A <dimension> of * may be used only with a table formal parameter.

If any <dimension> of a table formal parameter is specified as *, they all must be specified as *.

The number of words occupied by a table must not exceed MAXTABLESIZE.

2.1.2.2 TABLE STRUCTURE

Syntax:

```
<structure-specifier> ::= PARALLEL  
                        | T [<bits-per-entry>]  
  
<bits-per-entry> ::= <compile-time-integer-formula> (5.1.1)
```

Semantics:

Dimensioned tables can have a parallel or serial structure. In addition, a serial table may be tightly structured. The <structure-specifier> specifies the table structure.

A <structure-specifier> PARALLEL indicates parallel structure. For tables with parallel structure, the first word (word 0) of each entry is allocated consecutively, then word one, etc. An omitted <structure-specifier> or one with T indicates serial structure. For tables with a serial structure, all words of the first entry are allocated consecutively, then all words of the next entry, etc. Entries in both parallel and serial tables are arranged such that the rightmost indices vary fastest, from the lower bound to the upper bound.

A <structure-specifier> of T indicates tight structure (in addition to serial structure). Tight structure defines the allocation of storage between entries in a dimensioned (ordinary or specified) table, whereas packing (see Section 2.1.2.3) defines the allocation of storage within an entry of an ordinary table. Tight structure indicates that multiple entries of a dimensioned table are to be stored within a single word such that no entry crosses a word boundary. <Bits-per-entry> specifies the number of bits each entry is to occupy. If it is omitted, it will default to the minimum number of bits needed to store the entry.

Entries in tightly-structured tables are right-justified in the bits allotted.

Entries in tables without a <structure-specifier> of T shall not share a word.

Constraints:

<Bits-per-entry> must be equal to or greater than the minimum number of bits needed to store the entry.

The explicit or default value of <bits-per-entry> must be less than or equal to BITSINWORD.

Items in a parallel table must not cross word boundaries.

A parallel table must contain a <dimension-list>.

2.1.2.3 ORDINARY TABLE ENTRIES

Syntax:

<ordinary-entry-specifier>	::=	[<packing-specifier> <item-type-description> [<table-preset>] ;	(2.1.1) (2.1.6)
		[<packing-specifier> [<table-preset>] ; <ordinary-table-body>	(2.1.6)
<packing-specifier>	::=	N M D	
<ordinary-table-body>	::=	<ordinary-table-item-declaration> BEGIN <ordinary-table-options>... END	
<ordinary-table-item-declaration>	::=	ITEM <table-item-name> <item-type-description> [<packing-specifier> [<table-preset>] ;	(2.1.1) (2.1.6)
<table-item-name>	::=	<name>	(8.2.1)
<ordinary-table-options>	::=	<ordinary-table-item-declaration> <directive> <null-declaration>	(9.0) (2.7)

Semantics:

An `<ordinary-entry-specifier>` is used to specify the contents of an entry of an ordinary table.

No allocation order is implied by the order of items in the `<ordinary-table-options>` unless the `<ordinary-table-options>` contains an `<order-directive>`. If an `<order-directive>` is not in effect, `<ordinary-table-options>` will be reordered, if necessary, to reduce the storage occupied by an entry, consistent with the `<packing-specifier>`. Tables having the same type will have the same representation.

The `<packing-specifier>` specifies the density with which items are allocated within an entry. The following three degrees of packing can be specified:

1. N indicates that the items are not packed. No items share a word.
2. M indicates a density of packing that can be between N and D. The exact meaning is implementation-dependent, and is specified to be an effective compromise between space usage and accessing ease.
3. D indicates dense packing. Items are allocated adjacent bits in a word with the following exceptions:
 - a. Non-character items one word or longer start on a word boundary. Shorter non-character items do not cross word boundaries.
 - b. Each byte of a character item which crosses a word boundary must be allocated on a byte boundary. An implementation may (but need not) allocate the bytes of other character items on byte boundaries.

A `<packing-specifier>` preceding an `<ordinary-table-body>` in an `<ordinary-entry-specifier>` applies to all items in the `<ordinary-table-body>` that do not themselves include a `<packing-specifier>` in their declaration.

Default packing for a tightly-structured table (see Section 2.1.2.2) is D; for all other tables, it is N.

The value of unallocated bits in an `<ordinary-entry-specifier>` is implementation-dependent.

The `<table-preset>`, if present, specifies initial values for the table entries.

Constraints:

Only tables having STATIC allocation (implicitly or explicitly) may contain a <table-preset>.

Declaration of table <formal-input-parameters> or <formal-output-parameters> (see Section 3.3) must not contain <table-presets>.

If a <table-preset> precedes the <ordinary-table-body>, none of the <ordinary-table-item-declarations> in the <ordinary-table-body> can contain a <table-preset>.

An <ordinary-entry-specifier> used in a <table-type-declaration> (see Section 2.2) must not contain a <table-preset>.

An <ordinary-table-options> must contain at least one <ordinary-table-item-declaration>.

A <packing-specifier> of N is permitted in a tightly-structured table only if the table entry contains only one item.

The number of words allocated for a table entry must not exceed MAXBITS/BITSINWORD.

2.1.2.4 SPECIFIED TABLE ENTRIES

Syntax:

<specified-entry-specifier>	::=	<words-per-entry> <specified-item-description> [<table-preset>] ;	(2.1.6)
		<words-per-entry> [<table-preset>] ; <specified-table-body>	(2.1.6)
<words-per-entry>	::=	W [<entry-size>] V	
<entry-size>	::=	<compile-time-integer-formula>	(5.1.1)
<specified-item-description>	::=	<item-type-description> POS (<location-specifier>)	(2.1.1)

```

<location-specifier> ::= <starting-bit> ,
                        <starting-word>

<starting-bit>      ::= <compile-time-integer-formula>      (5.1.1)
                        | *

<starting-word>     ::= <compile-time-integer-formula>      (5.1.1)

<specified-table-body> ::= <specified-table-item-declaration>
                        | BEGIN
                          <specified-table-options>...
                          END

<specified-table-item-
  declaration>      ::= ITEM <table-item-name>              (2.1.2.3)
                        <specified-item-description>
                        [<table-preset>] ;                  (2.1.6)

<specified-table>   ::= <specified-table-item-declaration>
  options>
                        | <directive>                      (9.0)
                        | <null-declaration>                (2.7)

```

Semantics:

A <specified-entry-specifier> is used to specify the contents of an entry of a specified table.

<Words-per-entry> specifies the size of (i.e., number of words in) each entry in the table. <Words-per-entry> containing a W indicates a fixed-length-entry specified table whereas V indicates a variable-length-entry specified table. In a fixed-length-entry specified table, <entry-size> (if present) specifies the number of words allocated to each entry in the table. In a tightly-structured table (in which <entry-size> must be omitted), the size of the entry is determined from the <structure-specifier>. In a variable-length-entry specified table, each entry is allocated one word.

The <location-specifier> specifies the physical location of the item from the start of the entry. <Starting-word> indicates at which word of the entry, starting from zero, the item is to start, and <starting-bit> indicates at which bit in the word, starting from zero at the leftmost part of the word, the item is to start. In the case of entries in tightly structured tables, <starting-bit> is considered to be relative to the start of the entry. A <starting-bit> of * indicates

that the item should occupy the same amount of storage and be aligned in the same way it would if it were allocated outside a table, in order to ensure efficient access to the item. These rules apply to both fixed-length-entry and to variable-length-entry specified tables. Consequently, in a variable-length-entry table, reference to an item with subscript NN will reference that item relative to the start of the NNth entry, where each entry is considered to be one word long (i.e., a subscript of NN does not refer to the NNth logical entry in that table). It is entirely up to the programmer to keep track of the actual length of logical entries in such tables.

The <table-preset>, if present, specifies initial values for the table entries.

The value of unallocated bits in a <specified-table-entry> is implementation-dependent.

Constraints:

<Entry-size> must be omitted on tightly-structured tables and must be present otherwise.

<Entry-size> must be greater than zero and less than or equal to MAXBITS/BITSINWORD.

<Starting-word> must be non-negative. For items in tables with entry sizes specified by <entry-size>, <starting-word> plus number of words occupied by the item must not exceed <entry-size>. For tightly structured tables <starting-word> must be zero.

<Starting-bit> must be non-negative and must not cause item position to violate other positioning constraints. For non-tightly structured tables it must also be less than BITSINWORD. For tightly structured tables <starting-bit> plus number of bits occupied by the item must not exceed <bits-per-entry>.

Only tables having STATIC allocation (implicitly or explicitly) may contain a <table-preset>.

Tables that are <formal-input-parameters> or <formal-output-parameters> (see Section 3.3) must not contain a <table-preset>.

If a <table-preset> precedes the <specified-table-body>, none of the <specified-table-item-declarations> in the <specified-table-body> can contain a <table-preset>. If any part of an item in a <specified-table-body> overlaps any part of another item in the table body, only one of the items can be preset.

A <specified-entry-specifier> used in a <table-type-declaration> (see Section 2.2) must not contain a <table-preset>.

A <specified-table-options> must contain at least one <specified-table-item-declaration>.

Non-character items whose size is one word or less cannot cross a word boundary. Character items, regardless of length, may start on any byte boundary, i.e., any value of the machine parameter BYTEPOS. Any <starting-bit> value is permitted for character items that do not cross word boundaries.

An implementation may restrict legal <starting-bit> values for pointer items that are initialized.

Variable-length-entry specified tables must contain a <specified-table-body>, and they cannot contain <table-presets> or <structure-specifiers>.

2.1.3 CONSTANT DECLARATIONS

Syntax:

```
<constant-declaration> ::= CONSTANT ITEM
                           <constant-item-name>
                           <item-type-description>           (2.1.1)
                           <item-preset> ;                   (2.1.6)

                           | CONSTANT TABLE
                           <constant-table-name>
                           [<dimension-list>]                 (2.1.2.1)
                           <table-description>                 (2.1.2)

<constant-item-name>      ::= <name>                         (8.2.1)

<constant-table-name>     ::= <name>                         (8.2.1)
```

Semantics:

A <constant-declaration> creates an item or table whose value must be set by means of an <item-preset> or <table-preset> and whose value cannot be changed during execution of a program. The value of a constant item whose type class is not pointer can be used in a <compile-time-formula> (see Section 5.0). The value of a constant table or an item within a constant table may not be used in a <compile-time-formula>.

AD-A100 577

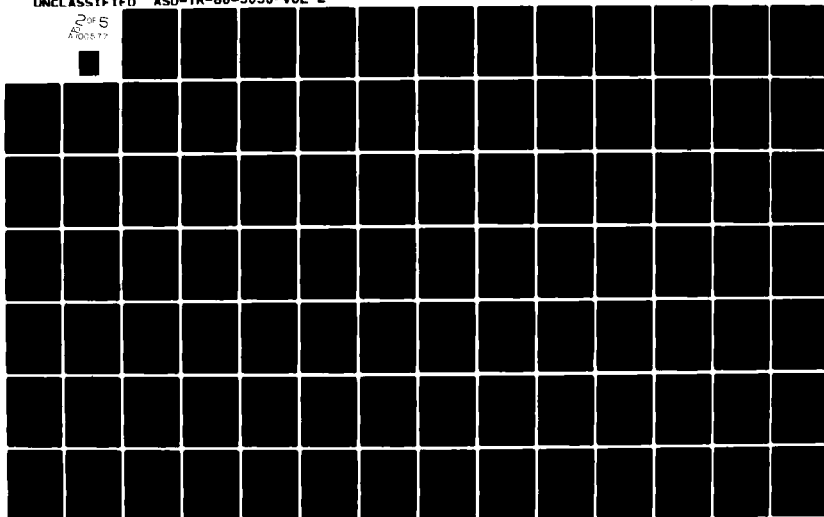
AERONAUTICAL SYSTEMS DIV WRIGHT-PATTERSON AFB OH
AFSC STANDARDIZATION CONFERENCE, 1553, 1589, 1750, 1760, ADA, N--ETC(U)
NOV 80 E C GANGL, S E SMITH
ASD-TR-80-5050-VOL-2

F/6 1/3

UNCLASSIFIED

NL

2 of 5
AD-A100 577



Physical storage will be allocated for all <constant-declarations> that are in <block-declarations>.

The allocation permanence of all allocated <constant-declarations> is considered to be STATIC, even if the declarations appear in a <subroutine-definition>.

2.1.4 BLOCK DECLARATIONS

Syntax:

<block-declaration>	::= BLOCK <block-name> [<allocation-specifier>] ; (2.1.5) <block-body-part>
	BLOCK <block-name> [<allocation-specifier>] (2.1.5) <block-type-name> (2.2) [<block-preset>] ; (2.1.6)
<block-name>	::= <name> (8.2.1)
<block-body-part>	::= <null-declaration> (2.7)
	<data-declaration> (2.1)
	BEGIN <block-body-options>... END (9.0)
<block-body-options>	::= <data-declaration> (2.1)
	<overlay-declaration> (2.6)
	<directive> (9.0)
	<null-declaration> (2.7)

Semantics:

A <block-declaration> declares a group of items, tables, and other blocks that are to be allocated in a contiguous area of storage. No allocation order is implied by the order of the declarations within a block unless the <block-body-options> contains an <order-directive>. If an <order-directive> is not in effect, <block-body-options> will be reordered, if necessary, to improve accessibility.

The <allocation-specifier> establishes the allocation permanence of blocks which are not enclosed in blocks. This allocation permanence is automatic if the declaration is in a subroutine and the <allocation-specifier> is omitted, otherwise it is STATIC (see Section 2.1.5). Blocks enclosed in blocks inherit the allocation permanence of the enclosing block.

The <block-declaration> describes the contents of the block either with a <block-type-name> (see Section 2.2) or with a <block-body-part>. The <block-body-part> contains explicit declarations of all the components of the block.

The <block-preset>, if present, specifies initial values for the block components. For <block-declarations> containing a <block-body-part> rather than a <block-type-name>, initial values may be specified with <block-presets>, <table-presets> and <item-presets> on the components themselves.

Constraints:

Only blocks having STATIC allocation (explicitly or by default) may contain a <block-preset> or a <data-declaration> containing an <item-preset>, <table-preset> or <block-preset>.

If a <constant-declaration> is in a block, the block must have STATIC allocation (explicitly or by default).

<Data-declarations> within a block must not contain an <allocation-specifier>.

Blocks that are <formal-input-parameters> or <formal-output-parameters> (see Section 3.3) must not contain an <allocation-specifier>, a <block-preset> or a <data-declaration> with an <item-preset>, <table-preset>, or <block-preset>.

Components of blocks declared with a <block-type-name> may be accessed only by using pointers to the blocks.

2.1.5 ALLOCATION OF DATA OBJECTS

Syntax:

<allocation-specifier> ::= STATIC

Semantics:

Allocation of storage for a data object can be STATIC or automatic. STATIC allocation means that the data object is to exist throughout the

entire execution of the program. Automatic allocation is applicable only to data declared within subroutines and means that the data object need only exist while the subroutine is executing (i.e., values are not necessarily preserved between calls). Automatic is the default allocation for data declared in subroutines and cannot be explicitly specified. STATIC is the default for data not declared in subroutines and can be explicitly specified both inside and outside of subroutines.

The treatment of STATIC data in a concurrent processing environment is implementation-dependent with respect to which data, if any, are shared among processes.

2.1.6 INITIALIZATION OF DATA OBJECTS

Syntax:

```

<item-preset> ::= = <item-preset-value>

<item-preset-value> ::= <compile-time-formula> (5.0)
                       | <loc-function> (6.3.1)

<table-preset> ::= = <table-preset-list>

<table-preset-list> ::= <default-preset-sublist>
                       | [<default-preset-sublist> ,]
                         <specified-preset-sublist>, ...

<default-preset-sublist> ::= <preset-values-option>, ...

<specified-preset-sublist> ::= <preset-index-specifier>
                              <preset-values-option>, ...

<preset-index-specifier> ::= POS ( <constant-index>, ... ) :

<constant-index> ::= <compile-time-integer-formula> (5.1.1)
                   | <compile-time-status-formula> (5.4)

<preset-values-option> ::= [<item-preset-value>]
                          | <repetition-count>
                            ( <preset-values-option>, ... )

<repetition-count> ::= <compile-time-integer-formula> (5.1.1)

```

```

<block-preset>          ::= = <block-preset-list>

<block-preset-list>     ::= <block-preset-values-option>, ...

<block-preset-values-   ::= <preset-values-option>
option>                  | [( <table-preset-list> )]
                           | [( <block-preset-list> )]
```

Semantics:

Items, tables, and blocks with STATIC allocation can be given initial values by means of <item-presets>, <table-presets>, and <block-presets>, respectively. Furthermore, constant items and tables must be given initial values with <item-presets> and <table-presets>. Initial values are values of the variables after a module has been loaded but prior to any dynamic reference to the variables. They do not imply any provision for later restoring values to the initial state.

An <item-preset> specifies an initial value for an item.

A <table-preset> specifies a list of initial values. If the <table-preset> occurs on an item within an entry of a table, the <table-preset> specifies values only for that item. If the table is dimensioned, the <table-preset> for the item, if present, may specify a list of values to initialize that item in each entry of the dimensioned table.

If the <table-preset> occurs on an entry of a table, the <table-preset> specifies values for all items within that entry. If the table is dimensioned, the <table-preset> specifies values for all the items in each entry of the dimensioned table. Assuming the entry has N items in it, the first N values in the <table-preset> are initial values for the N items in the first entry of the table (in the order in which the declarations appear), the second N values in the <table-preset> are initial values for the N items in the second entry of the table, etc.

Entries within a dimensioned table are normally initialized in order, the first entry being the one with the lowest value of each dimension index, and proceeding with the rightmost indices increasing most rapidly. This is the procedure followed when a <default-preset-sublist> is specified. If a <specified-preset-sublist> is used, initialization using the values in the sublist will start with the entry whose indices are specified in the <preset-index-specifier> and will proceed with the rightmost indices increasing most rapidly.

A <repetition-count> can be used as a shorthand to specify the number of consecutive repetitions of the sequence of <preset-values-

options> enclosed in the parentheses following the <repetition-count>.

If a value is omitted in the <table-preset>, the item corresponding to the omitted value will remain uninitialized and cannot be given an initial value elsewhere in the preset.

A <block-preset> is used only to initialize a block declared with a <block-type-name>. The <block-preset-list> specifies initial values for the items, tables and blocks contained within the block in the order of their declaration. A parenthesized <table-preset-list> is used to initialize a contained table and a parenthesized <block-preset-list> is used to initialize a contained block. An omitted entry from the list indicates that the corresponding item, table, or block will remain uninitialized.

Constraints:

The type of each value in an <item-preset>, <table-preset> or <block-preset> must match or be implicitly convertible to the type of the data object being initialized (see Section 7.0).

The <preset-index-specifiers> within a <table-preset> must be specified such that no bit position is initialized more than once and the bounds of the table are not exceeded.

The value of the <repetition-count> must not be negative.

An item must not be initialized more than once by initializing another item that overlaps it.

The type of each <constant-index> in a <preset-index-specifier> must match the type of the bounds of the corresponding dimension in the <dimension-list> of the declaration of the table.

The number of <constant-indices> in a <preset-index-specifier> must be the same as the number of <dimensions> in the table's <dimension-list>.

If the argument of a <loc-function> used as a preset value is a <named-variable>, it must be a <data-name> for an object whose allocation permanence is STATIC, either explicitly or by default.

2.2 TYPE DECLARATIONS

Syntax:

<type-declaration> ::= <item-type-declaration>
 | <table-type-declaration>
 | <block-type-declaration>

<item-type-declaration> ::= TYPE <item-type-name>
 <item-type-description> ; (2.1.1)

<item-type-name> ::= <name> (8.2.1)

<table-type-declaration> ::= TYPE <table-type-name>
 TABLE <table-type-specifier>

<table-type-specifier> ::= [<dimension-list>] (2.1.2.1)
 [<structure-specifier>] (2.1.2.2)
 [<like-option>]
 <entry-specifier> (2.1.2)

 | [<dimension-list>] (2.1.2.1)
 <table-type-name> ;

<table-type-name> ::= <name> (8.2.1)

<like-option> ::= LIKE <table-type-name>

<block-type-declaration> ::= TYPE <block-type-name>
 BLOCK <block-body-part> (2.1.4)

<block-type-name> ::= <name> (8.2.1)

Semantics:

A <type-declaration> is used to give a name to a type specification.

An <item-type-declaration> associates the <item-type-name> with the <item-type-description>.

A <table-type-declaration> associates the <table-type-name> with the <table-type-specifier>.

If a <like-option> is specified, the entry being described consists of the items in the type named in the <like-option> together with the items in the <entry-specifier>. The physical positioning of items in

the <like-option> relative to the start of the entry is fixed at the time the <like-option> type name is declared and is not changed by its use as a <like-option>. If the type named in the <like-option> contains a <dimension-specifier>, it applies to the entire <table-type-specifier>. If the table is an ordinary table, the <packing-specifier>, if present, only applies to the items in the <entry-specifier>, not to the items obtained from the <like-option>. If the table is a specified table, the <words-per-entry> in the <entry-specifier>, if present, specifies the total size of the entry including the items obtained from the <like-option>. If the <table-type-declaration> contains a <structure-specifier> of T, <bits-per-entry> specifies the total number of bits the entry is to occupy including items obtained from the <like-option>. If <bits-per-entry> is omitted it will default to the minimum number of bits needed to store the entry, including items obtained from the <like-option>.

The physical representation of a table type is fixed by the type declaration. All objects allocated with such a type name will have the same representation. In particular, the position of table items in a <like-option> is not modified by the occurrence of a <packing-specifier> or <order-directive> in the <entry-specifier>. However, unused space in the portion described by the <like-option> can be occupied by table items given in a packed <entry-specifier>.

A <block-type-declaration> associates the <block-type-name> with the <block-body-part>.

For type matching purposes, a type name is considered to be an abbreviation for its associated <item-type-description>, <table-type-specifier>, or <block-body-part>, in any context except within a <pointer-item-description>.

Constraints:

The <item-type-description>, <table-type-specifier>, or <block-body-part> in a <type-declaration> must not contain an <item-preset>, <table-preset>, or <block-preset>.

A <block-body-part> in a <block-type-declaration> cannot contain <constant-declarations>.

If a <table-type-specifier> contains a <dimension-list>, then it must not contain a <table-type-name> (either directly or in a <like-option>) whose <table-type-declaration> contains a <dimension-list>.

Tables may be characterized as parallel, serial, tight, ordinary, variable-length-entry, and specified. The characterizations of the table type in a <like-option> must be the same as those of the <table-type-declaration> in which the <like-option> appears.

<Words-per-entry> of the <table-type-specifier> must not specify a value that is less than <words-per-entry> of the type name specified in a <like-option>.

The (explicit or default) number of bits per entry in a <table-type-specifier> having tight structure must not be less than the number of bits per entry of the type name specified in a <like-option>.

A `<table-type-name>` must be a `<name>` declared in a `<table-type-declaration>`.

A `<block-type-name>` must be a `<name>` declared in a `<block-type-declaration>`.

Note:

A `<table-type-name>`, `<item-type-name>`, or `<block-type-name>` must not be a formal parameter name or an actual parameter name.

2.3 STATEMENT NAME DECLARATIONS

Syntax:

$$\langle \text{statement-name-declaration} \rangle ::= \text{LABEL} \quad \langle \text{statement-name} \rangle, \dots ; \quad (4.0)$$

Semantics:

A `<statement-name-declaration>` is used to explicitly declare a `<statement-name>`. Ordinarily, a `<statement-name>` is implicitly declared by its use in a `<label>`. An explicit `<statement-name-declaration>`, however, must be used for statement name `<formal-input-parameters>`, for statement names that are the same as `<define-names>` declared in an enclosing scope, and for external `<statement-name-declarations>`.

Constraints:

The `<statement-names>` in a `<statement-name-declaration>` must either be `<formal-input-parameters>` to the subroutine containing the `<statement-name-declaration>` or else must be used in `<labels>` in the immediate scope containing the `<statement-name-declaration>` (i.e., not including nested scopes).

2.4 DEFINE DECLARATIONS

Syntax:

<code><define-declaration></code>	<code>::=</code>	<code>DEFINE <define-name></code> <code><definition-part></code>	
<code><define-name></code>	<code>::=</code>	<code><name></code>	(8.2.1)
<code><definition-part></code>	<code>::=</code>	<code>[<formal-define-parameter-list></code> <code><define-string> ;</code>	
<code><formal-define-parameter-</code> <code>list></code>	<code>::=</code>	<code>(<formal-define-parameter>,...)</code>	
<code><formal-define-parameter></code>	<code>::=</code>	<code><letter></code>	(8.1)
<code><define-string></code>	<code>::=</code>	<code>" [<character>...] "</code>	(8.1)

Semantics:

A `<define-declaration>` is used to associate a name with a (possibly parameterized) text string, the `<define-string>`. The `<define-string>` will be substituted for the `<define-name>` when the `<define-name>` is used in a `<define-call>` (see Section 2.4.1).

The `<formal-define-parameter-list>` is used to declare `<formal-define-parameters>`. These parameters receive values from `<actual-define-parameters>` in each `<define-call>` (see Section 2.4.1). The values are substituted in the `<define-string>` wherever the `<formal-define-parameters>` are referenced. Reference to a `<formal-define-parameter>` within the `<define-string>` is indicated by preceding the parameter name with an exclamation point. Such parameter references can occur anywhere within the `<define-string>` and, by appropriate juxtaposition, can be used to create new symbols.

Within the `<define-string>`, the quotation mark (") and exclamation point (!) can be used as simple characters by doubling them. A `<define-string>` is terminated by the first undoubled quotation mark, regardless of the lexical context in which the undoubled quotation mark appears.

As with other `<names>`, a `<define-name>` is known in the scope containing its declaration and may be redeclared in an inner scope.

The `<define-string>` may contain `<define-calls>`. Such calls will be expanded for each substitution of the `<define-string>`, using the definition active in the scope of the `<define-call>`.

Constraints:

A <comment> delimited by quotation marks must not occur in a <define-declaration> between the <define-name> and the <define-string>.

Circular <define-declarations> as the result of <define-strings> containing <define-calls> are not allowed.

The same <letter> must not appear more than once in any <formal-define-parameter-list>.

2.4.1 DEFINE CALLS

Syntax:

<define-call> . ::= <define-name> (2.4)
[<actual-define-parameter-list>]

<actual-define-parameter- ::= (<actual-define-parameter>,...)
list>

<actual-define-parameter> ::= [<character>...] (8.1)

| " [<character>...] " (8.1)

Semantics:

A <define-call> is used to cause textual substitution to occur. A <define-call> is processed as follows:

1. The characters comprising <actual-define-parameters> are substituted for the corresponding <formal-define-parameters> in the <define-string> associated with the <define-name>.
2. The resulting <define-string> logically replaces the <define-call>.
3. The substituted <define-string> is scanned from its beginning to determine what <symbols> it contains; these <symbols> are processed as though they had appeared in the original text at the point of the replaced <define-call>.

Note that the substituted source text may be found to contain <define-calls> and these are processed in the same manner.

If an <actual-define-parameter> is omitted, a null string will be substituted for the <formal-define-parameter>. If the number of <formal-define-parameters> exceeds the number of <actual-define-parameters>, null strings will be substituted for the

MIL-STD-1589B (USAF)
06 June 1980

trailing <formal-define-parameters>.

If an <actual-define-parameter> consists of characters enclosed in quotation marks, all the enclosed characters are substituted. The quotation mark (") must be doubled within an actual parameter enclosed in quotes.

If an <actual-define-parameter> does not contain enclosing quotation marks, the characters substituted are the first non-blank character and subsequent characters ending at, but not including, either (1) the first right parenthesis that is not balanced by a left parenthesis that is part of the <actual-define-parameter>, or (2) the first comma that is not between such balanced parentheses.

<Define-calls> are not recognized in <comments> and <character-literals>.

Constraints:

The <actual-define-parameter-list> must not be omitted if the corresponding <define-declaration> contains a <formal-define-parameter-list>.

The number of <actual-define-parameters> must not be greater than the corresponding number of <formal-define-parameters>.

A <define-call> cannot be juxtaposed with surrounding symbols so as to create new symbols after substitution.

A <define-call> must not be used as the <name> being declared within a declaration.

A <define-call> must not be used as a <formal-input-parameter> or <formal-output-parameter> within a <procedure-heading> or <function-heading>.

The same <letter> must not appear more than once in any <formal-define-parameter-list>.

Note:

The define-listing directives (see Section 9.7.2) allow programmer control over whether the source program listing contains the expanded string, the define invocation, or both, for <define-calls>.

Examples:

```
DEFINE FOO(A) "BAZ !AFAZ !A";  
DEFINE BAR "HELLO";
```

```
DEFINE BARFAZ "GOODBYE";  
DEFINE HELLOFAZ "NOT USED";  
FOO(BAR)
```

The result of the <define-call>, FOO(BAR), after substituting for the formal parameter !A is BAZ BARFAZ BAR, and after rescanning this string, the final result is BAZ GOODBYE HELLO.

```
DEFINE MAKEDEF(N, S) "DEFINE IN ""!S""";  
MAKEDEF(NEW, HELLO);
```

The result is DEFINE NEW "HELLO"; , i.e., a new <define-declaration>.

2.5 EXTERNAL DECLARATIONS

Syntax:

<external-declaration> ::= <def-specification> (2.5.1)

| <ref-specification> (2.5.2)

Semantics:

<External-declarations> declare <names> that are potentially known in other <modules> of the <complete-program>. Such names are said to be external.

Constraint:

Formal parameter names cannot be declared external.

2.5.1 DEF SPECIFICATIONS

Syntax:

<def-specification> ::= <simple-def>

| <compound-def>

<simple-def> ::= DEF
<def-specification-choice>

<compound-def> ::= DEF BEGIN
<def-specification-choice>...
END

```
<def-specification-choice> ::= <null-declaration>           (2.7)
                             | <data-declaration>           (2.1)
                             | <def-block-instantiation>
                             | <statement-name-declaration> (2.3)
                             | <directive>                 (9.0)
                             | <def-specification-choice>

<def-block-instantiation> ::= BLOCK INSTANCE
                             <block-name> ;                (2.1.4)
```

Semantics:

<Def-specifications> enable data objects to be declared that are potentially available via <ref-specifications> and/or <compool-directives> for use in other <modules>. Physical storage will be allocated for these objects.

Either a <def-block-instantiation> or a <block-declaration> may be used in a <def-specification> to create a block with external scope. However, in order for a <def-block-instantiation> to be meaningful, a <ref-specification> containing a <block-declaration> having the same <block-name> must exist, either in that <module> or in a <compool-module> that is referenced via a <compool-directive>. Preset information used in the creation of a block declared with a <def-block-instantiation> will be obtained from the corresponding <ref-specification>.

A <statement-name-declaration> in a <def-specification> makes the addresses of the designated statements available as linkage information in the environment of the <complete-program> but does not make these names available as targets of out-of-scope GOTO statements (see Section 4.7).

Constraints:

A data declaration in a <def-specification> and a corresponding declaration in a <ref-specification> must agree in name, type, and all attributes. However, a compiler will perform this check across <module> boundaries only if a connection is established between the <modules> via a <compool-directive>.

External data must have STATIC allocation, either explicitly or implicitly.

The <data-declaration> in a <def-specification> cannot be a <constant-declaration>. (This constraint does not prevent <constant-declarations> from appearing in <block-declarations> in <def-specifications>.)

2.5.2 REF SPECIFICATIONS

Syntax:

```

<ref-specification>      ::= <simple-ref>
                           | <compound-ref>

<simple-ref>              ::= REF
                           <ref-specification-choice>

<compound-ref>           ::= REF BEGIN
                           <ref-specification-choice>...
                           END

<ref-specification-choice> ::= <null-declaration>          (2.7)
                           | <data-declaration>           (2.1)
                           | <subroutine-declaration>      (3.0)
                           | <directive>                  (9.0)
                           <ref-specification-choice>
                           | <statement-name-declaration> (2.3)

```

Semantics:

A <ref-specification> enables a <module> to reference a <name> whose <def-specification> is in another <module>.

Physical storage for external names occurs in the <module> containing the <def-specification> and not in the <module> containing the <ref-specification>.

A <ref-specification> for a <name> may appear in a <compool-module> and the corresponding DEF in another <module>. In that case, the <name> will be available for use in any other module of the <complete-program>, provided that the referencing module has the appropriate <compool-directive>. The compiler will enforce the requirement that the DEF and the REF specifications agree, provided the <module> containing the DEF has the appropriate <compool-directive>. Alternatively, the <ref-specification> may appear in the accessing <module> directly

instead of in a compool (bypassing the compool entirely), but in this case it will be beyond the compiler's ability to check for compatibility between the DEF and the REF specifications. When <ref-specifications> are used outside of compools to gain access to external names, the programmer is entirely responsible for the correct usage of those names.

For <data-declarations> in a <def-specification> that is in a <compool-module>, no <ref-specification> is necessary if the accessing module has a <compool-directive> that causes that data to be imported (see Section 9.1).

Constraints:

For every <data-declaration> in a <ref-specification>, there must exist a corresponding declaration in a <def-specification>. For every <subroutine-declaration> in a <ref-specification>, there must exist in some <procedure-module> or <main-program-module> a corresponding <procedure-definition> preceded by DEF.

In a <ref-specification>, presets are illegal in <item-declarations> and <table-declarations> and are optional in <block-declarations>. A <ref-specification> that contains presets can be used only in conjunction with a <def-block-instantiation>.

A <data-declaration> in a <ref-specification> cannot be a <constant-declaration>. (This constraint does not prevent <constant-declarations> from appearing in <block-declarations> in <ref-specifications>).

2.6 OVERLAY DECLARATIONS

Syntax:

```
<overlay-declaration> ::= OVERLAY
                        [<absolute-address>]
                        <overlay-expression> ;

<absolute-address>    ::= POS ( <overlay-address> ) ;

<overlay-address>     ::= <compile-time-integer-formula> (5.1.1)

<overlay-expression> ::= <overlay-string>:...

<overlay-string>      ::= <overlay-element>,...

<overlay-element>     ::= <spacer>
```

		<data-name>	
		(<overlay-expression>)	
<spacer>	::=	W <compile-time-integer-formula>	(5.1.1)
<data-name>	::=	<item-name>	(2.1.1)
		<table-name>	(2.1.2)
		<block-name>	(2.1.4)

Semantics:

An <overlay-declaration> is used to specify any or all of the following:

- 1) that data objects are to have a specific allocation order
- 2) that certain data objects are to occupy the same memory locations as other data objects
- 3) that certain data objects are to be allocated at a particular absolute memory location

The <overlay-elements> in an <overlay-string> are allocated memory locations in the order of their appearance in the string. The memory locations allocated to the elements of an <overlay-string> that appears to the left of a colon in an <overlay-expression> are overlaid with the space allocated to the elements of the <overlay-string> that appears to the right of the colon.

The <overlay-address> specifies an absolute memory location at which allocation of the <overlay-expression> begins. The meaning of an overlay address is machine dependent.

A <spacer> in an <overlay-string> specifies a number of words to be skipped during allocation.

Constraints:

The allocation permanence of all data objects in an <overlay-declaration> must be the same (i.e., all STATIC or all automatic).

An <overlay-declaration> within a <block-declaration> or <block-type-declaration> must not reference data names declared outside the block or block type or within nested blocks.

An <overlay-declaration> outside a <block-declaration> or <block-type-declaration> must not reference data names declared within a block or block type.

An <overlay-declaration> within a <block-declaration> or <block-type-declaration> must not include an <absolute-address>.

A <block-declaration> or <block-type-declaration> must not include an <overlay-declaration> if an <order-directive> is in effect for the block or block type.

Declarations for all <data-names> in an <overlay-declaration> must precede the <overlay-declaration>, and all must be in the same scope.

<Overlay-declarations> cannot be used to specify more than one physical location to any data object.

Names of formal parameters cannot be used in <overlay-declarations>.

If an <overlay-address> is specified, all <data-names> used in the <overlay-expression> must have an (explicit or default) allocation permanence of STATIC.

Note:

A <data-name> in an <overlay-declaration> cannot be declared in a <constant-declaration>.

2.7 NULL DECLARATIONS

Syntax:

```
<null-declaration>      ::= ;  
                           | BEGIN END
```

Semantics:

A <null-declaration> has no semantic effect.

06 June 1980

3.0 PROCEDURES AND FUNCTIONS

Syntax:

<subroutine-declaration> ::= <procedure-declaration> (3.1)

| <function-declaration> (3.2)

<subroutine-definition> ::= <procedure-definition> (3.1)

| <function-definition> (3.2)

| <directive> (9.0)

<subroutine-definition>

Semantics:

Subroutines describe algorithms that may be executed from more than one place in a <complete-program>. A subroutine is either a procedure, which is invoked by a <procedure-call-statement>, or a function, which is invoked by a <function-call>.

A <subroutine-definition> contains the executable code for the subroutine, in addition to declarations for all local data and formal parameters, as well as definitions of any nested subroutines. A <subroutine-definition> is said to define the subroutine.

A <subroutine-declaration>, on the other hand, is said to declare the subroutine. A <subroutine-declaration> contains the heading of the subroutine and <declarations> for the formal parameters, but it contains no executable code. A <subroutine-declaration> is required in a <ref-specification> for each subroutine that is invoked in a module other than the module containing its definition (see Section 2.5). A <subroutine-declaration> is also required in two other situations: (1) when a subroutine name is declared as a formal parameter (see Section 3.3), and (2) when the name of the subroutine is the same as a <define-name> in an enclosing scope. It is not necessary to provide a <subroutine-declaration> if the subroutine is invoked only in the <module> where it is defined and if its name is not passed as a parameter or used in an enclosing scope as a <define-name>.

Constraints:

The <procedure-heading> or <function-heading> of a <subroutine-declaration> and that of the corresponding <subroutine-definition> must have identical attributes, and the parameters (both input and output) must agree in number, type, and order. (This constraint will be enforced only when the declaration is known in the scope of the definition.) Also, the subroutine name in the declaration and

definition must be the same (unless the <subroutine-declaration> is for a formal parameter).

3.1 PROCEDURES

Syntax:

<procedure-declaration>	::= <procedure-heading> ; <declaration>	(2.0)
<procedure-definition>	::= <procedure-heading> ; [<directive>...] <procedure-body>	(9.0)
<procedure-heading>	::= PROC <procedure-name> [<subroutine-attribute>] [<formal-parameter-list>]	(3.3)
<subroutine-attribute>	::= REC RENT	
<procedure-name>	::= <name>	(8.2.1)
<procedure-body>	::= <subroutine-body>	
<subroutine-body>	::= <statement>	(4.0)
	BEGIN [<declaration>...] <statement>... [<subroutine-definition>...] [<directive>...] [<label>...] END	(2.0) (4.0) (3.0) (9.0) (4.0)

Semantics:

The <procedure-heading> in a <procedure-declaration> may contain a <formal-parameter-list>, which specifies the names that are used in the <procedure-body> to refer to the corresponding arguments supplied by each call of the procedure. The syntax, semantics, and constraints for a procedure's <formal-parameter-list> are the same as for a function's <formal-parameter-list>, and are presented in Section 3.3.

The differences between a <procedure-declaration> and a <procedure-definition> are described in Section 3.0.

A <subroutine-attribute> of REC indicates that the subroutine is potentially recursive, i.e., that at run time, an invocation of the

subroutine may be dynamically nested within another invocation of it. If REC is present, physical allocation of locally-declared automatic data will occur dynamically. The data will be allocated and deallocated when the subroutine is entered and exited, respectively. This assures that separate copies of the local data will exist for each successive call in the recursive chain. Locally-declared STATIC data, however, will be allocated once, and the same storage will be used for all calls of that subroutine throughout the <complete-program>.

A <subroutine-attribute> of RENT indicates that the subroutine is re-entrant and may therefore be executed concurrently in a concurrent processing environment. A recursive subroutine is also re-entrant.

If execution of the <procedure-body> is completed without executing a RETURN statement, an ABORT statement, or a GOTO statement whose target is the name of a formal parameter, an implicit RETURN statement is executed.

Constraints:

A `<procedure-declaration>` can contain no `<declarations>` other than those for the procedure's formal parameters. `<Declarations>` of local data appear only in the procedure's definition.

A procedure must not be invoked recursively if it is not declared
REC.

A procedure must not be invoked re-entrantly if it is not declared RENT or REC.

A <subroutine-body> must contain at least one non-null <statement> (e.g., RETURN).

3.2 FUNCTIONS

Syntax:

<function-declaration>	::=	<function-heading> ; <declaration>	(2.0)
<function-definition>	::=	<function-heading> ; {<directive>...} <function-body>	(9.0)
<function-heading>	::=	PROC <function-name> [<subroutine-attribute>]	(3.1)
		[<formal-parameter-list>]	(3.3)
		<item-type-description>	(2.1.1)

<function-name> ::= <name> (8.2.1)

<function-body> ::= <subroutine-body> (3.1)

Semantics:

The differences between a <function-declaration> and a <function-definition> are described in Section 3.0.

The <item-type-description> specifies the type of the return value of the function. Within the body of the function, the name of the function may be assigned to as a variable in an <assignment-statement>. When the function is exited, the most recent value assigned to the <function-name> is used as the value of the function. The return value is considered to be allocated as automatic storage (see Section 2.1.5).

Use of the <function-name> in a <formula> within the body of the function is a recursive invocation of the function. Within the body of the function, the <function-name> may also be used as an <actual-input-parameter> in a subroutine call when the corresponding <formal-input-parameter> is a <function-name> (see Section 3.3).

The <function-heading> in a <function-declaration> or <function-definition> may contain a <formal-parameter-list>, which specifies the names that are used in the <function-body> to refer to the corresponding arguments supplied by each call of the function. The syntax, semantics, and constraints for a function's <formal-parameter-list> are the same as for a procedure's <formal-parameter-list>, and are presented in Section 3.3.

The inclusion of an <item-type-description> in the heading of a subroutine indicates that the subroutine is a function.

The <subroutine-attributes> of REC and RENT apply to functions in the same way as for procedures (see Section 3.1).

If execution of the <function-body> is completed without executing a RETURN statement, an ABORT statement, or a GOTO statement whose target is the name of a formal parameter, an implicit RETURN statement is executed.

Constraints:

The <function-name> may not be used as an <actual-output-parameter>.

The <function-name> is not declarable as a <name> within the function body.

A <function-declaration> can contain no <declarations> other than those for the functions formal parameters. <Declarations> of local data appear only in the function's definition.

A function must not be invoked recursively if it is not declared REC.

A function must not be invoked re-entrantly if it is not declared RENT or REC.

The <function-name> must be assigned a value before the function is exited.

3.3 PARAMETERS OF PROCEDURES AND FUNCTIONS

Syntax:

```
<formal-parameter-list> ::= ( [ <formal-input-parameter>, ... ]  
                               [ : <formal-output-parameter>, ... ] )  
  
<formal-input-parameter> ::= [ <parameter-binding> ]  
                               <input-parameter-name>  
  
<formal-output-parameter> ::= [ <parameter-binding> ]  
                               <output-parameter-name>  
  
<parameter-binding>      ::= BYVAL  
                               | BYREF  
                               | BYRES  
  
<input-parameter-name>   ::= <data-name> (2.6)  
                               | <statement-name> (4.0)  
                               | <subroutine-name>  
  
<output-parameter-name>  ::= <data-name> (2.6)  
  
<subroutine-name>        ::= <procedure-name>  
                               | <function-name> (3.2)
```

Semantics:

Parameters permit subroutines to have locally-declared <names> that correspond to entities whose values can be different for different

calls.

<Formal-input-parameters> and <formal-output-parameters> constitute the formal parameters of the subroutine. When the subroutine is invoked, the formal parameters are associated with a corresponding list of actual parameters supplied in the subroutine call (see Section 4.5).

<Formal-input-parameters> transfer values into the <subroutine-body> from the corresponding <actual-input-parameters>. <Formal-output-parameters> transfer values into the <subroutine-body> and also transfer values from the <subroutine-body> back to the corresponding <actual-output-parameters>.

If a formal parameter is a <data-name> it may be bound to the corresponding actual parameter in any of the following ways: by reference, by value, by result, or by value-result. Reference binding means that the actual parameter and the formal parameter denote the same physical object. Any change in the value of the formal parameter entails an immediate change in the value of the actual parameter and vice-versa. Value-result binding means that the formal parameter denotes a separate data object, assigned the value of the actual parameter on entry to the subroutine, and used to assign its value to the actual parameter on normal exit from the subroutine. Since it is a separate data object there is no interaction between it and the actual parameter during execution of the subroutine. Value binding is similar except the actual parameter is not modified on exit from the subroutine. Result binding leaves the value of the formal parameter undefined on entry to the subroutine but is otherwise like value-result binding.

Standard rules for types of binding indicate the effect normally required:

Reference binding shall be used for blocks, tables, and for entries of all except tight tables.

Value binding shall be used for input items and tight table entries.

Value-result binding shall be used for output items and tight table entries.

Explicit <parameter-binding> specification affects these rules as follows:

BYREF - reference binding is required. If the actual parameter cannot be passed by reference (such as a badly aligned table item), the compiler shall allocate a temporary

variable, use value or value-result binding as appropriate to pass the parameter between its actual location and the temporary variable, and pass the temporary variable by reference to the subroutine.

BYVAL - reference is prohibited. The parameter shall be passed by value or value-result as appropriate.

BYRES - result binding is required.

An implementation may optimize binding methods provided it guarantees required results both in parameters passed and in side effects, if any.

If the <formal-input-parameter> is a <statement-name>, a GOTO statement with that name as a target will cause the subroutine to be exited without setting any of the value-result parameters. Execution will resume at the statement named in the actual parameter as though the GOTO statement had been executed at the point of the subroutine call.

If the <formal-input-parameter> is a <subroutine-name> the <name> of the corresponding actual parameter determines which <subroutine-definition> to associate with the formal parameter's <subroutine-declaration> on each call. A call to that subroutine via the formal parameter <name> will be treated as if the corresponding actual parameter subroutine had been called from the same environment in which <subroutine-name> was originally specified as an <actual-input-parameter>.

The order of evaluation of actual parameters is unspecified.

In the absence of an <interference-directive>, no interference is assumed within the subroutine between actual parameter data and formal table or block parameters, or between actual parameters and variables accessed directly from within the subroutine.

Constraints:

The same name must not appear more than once in any <formal-parameter-list>.

A <formal-input-parameter> cannot be used in a context in which its value can be altered (e.g., as a target in an <assignment-statement>).

Names of data declared as formal parameters must not be used in <overlay-declarations>.

Declarations of formal parameters must not contain <allocation-specifiers> or presets.

<External-declarations> of formal parameters are not permitted.

The <subroutine-definition> (and <subroutine-declaration>, if one is present) must contain an explicit <declaration> for each <name> in the <formal-parameter-list>.

For any subroutine call, the number of formal and actual input parameters must be the same, and the number of formal and actual output parameters must be the same.

Declarations of formal parameters cannot be <constant-declarations> or <type-declarations>.

For all table parameters, the types of the formal parameters and those of the corresponding actual parameters must be equivalent (see Section 7.0). This requirement extends to the types and associated attributes of all components, and their allocation order. For all item parameters, the rules for implicit attribute conversion apply (see Section 7.0). Block parameters match under the following conditions: (1) the type and textual order of the components match exactly; (2) an !ORDER directive is either present in both <block-body-parts> or absent in both <block-body-parts>; and (3) <overlay-declarations> in both blocks have the same effect.

The actual parameter corresponding to a formal input parameter <statement-name> must be a <statement-name>. The actual parameter corresponding to a formal input parameter <subroutine-name> must be the name of a subroutine. Parameter types and return value types of formal and actual subroutines must match exactly.

BYRES binding must not be specified for input parameters.

3.4 INLINE PROCEDURES AND FUNCTIONS

Syntax:

```
<inline-declaration>      ::=  INLINE  
                             <subroutine-name>,... ;      (3.1)
```

Semantics:

An <inline-declaration> causes the object code for the bodies of each of the designated subroutines to be inserted at the point of every call of that subroutine within the scope containing the <inline-declaration>. This will be done instead of inserting code for calling a remote subroutine body.

The effect of the <inline-declaration> extends for just the name scope in which the <inline-declaration> appears. It does not affect calls appearing in enclosing scopes.

If any actual parameters to inline subroutines are constants, inline expansion may cause some formulas in the <subroutine-bodies> to become evaluable at compile time. Compile-time evaluation of these formulas will be performed and any corresponding error messages will be generated as though the programmer had written those formulas directly. Except for the effects of compile-time evaluation, the semantics of inline subroutine expansion are identical to the semantics of the normal, remote subroutine call mechanism.

Inline subroutines may themselves contain (possibly inline) subroutine calls, but they may not contain nested subroutine definitions.

Inline subroutine names may be used as actual parameters, but a call to the matching formal parameter name will result in a closed rather than inline invocation (even if the actual parameter is an inline subroutine).

Constraints:

Names of subroutines whose definitions appear in another module cannot be used in <inline-declarations>.

Formal parameters cannot be declared to be inline.

It is illegal to have an inline subroutine invocation of a subroutine that is already being expanded inline.

Formal parameters of inline subroutines cannot be used in contexts where the syntax requires a compile-time formula.

3.5 MACHINE-SPECIFIC PROCEDURES AND FUNCTIONS

Semantics:

Each compiler implementation may provide a set of procedures and functions that are intrinsically recognized by the compiler. These procedures and functions shall typically encompass operations that are not directly provided by the language. They may be implemented as subroutines or via inline code, whichever is suitable. The use of inline code is particularly suitable as a vehicle for invoking single machine instructions which are peculiar to the target machine.

MIL-STD-1589B (USAF)
06 June 1980

In general, a subroutine will be provided for machine instructions whose execution would otherwise be unobtainable through the language. It is not intended that every target machine instruction be supported as a machine-specific procedure or function. Subroutines will, however, be provided for machine-specific instructions whose meaning is not expressible in the language (e.g., "load status word", "test condition code"), as well as instructions for which a J73 subroutine could be written but which are directly implemented by target-machine instructions (e.g., "sine", "matrix multiply", or "rotate length-32 bitstring", etc.). Such subroutines will be defined at system scope and hence their names will be redefinable in inner scopes. Such subroutines will be invoked in the same way as other subroutines (see Sections 4.5 and 6.3). The particular parameters to such subroutines are subroutine-dependent.

Implementation requirements for each such subroutine include specification of the operation to be performed and of the rules for each formal parameter, including both its JOVIAL attributes and how it is used. The compiler shall generate code to use the parameters and perform the specified operation.

4.0 STATEMENTS

Syntax:

<statement>	::=	[<directive>...] [<label>...] <simple-statement>	(9.0)
		[<directive>...] [<label>...] <compound-statement>	(9.0)
<simple-statement>	::=	<assignment-statement>	(4.1)
		<loop-statement>	(4.2)
		<if-statement>	(4.3)
		<case-statement>	(4.4)
		<procedure-call-statement>	(4.5)
		<return-statement>	(4.6)
		<goto-statement>	(4.7)
		<exit-statement>	(4.8)
		<stop-statement>	(4.9)
		<abort-statement>	(4.10)
		<null-statement>	
<null-statement>	::=	;	
		BEGIN [<label>...] END	
<label>	::=	<statement-name> :	
<statement-name>	::=	<name>	(8.2.1)
<compound-statement>	::=	BEGIN <statement>... [<directive>...] [<label>...] END	(9.0)

<Statements> are the means by which computational algorithms are specified. They control the execution of the <complete-program>.

A **<compound-statement>** permits a sequence of **<statements>** to be used in contexts requiring a single **<statement>**.

A <null-statement> results in no operation.

A <label> is used to attach a <statement-name> to a <statement>. A <label> that is attached to the END of a <compound-statement> or <null-statement> is treated as if a no operation <statement> followed the <label>.

4.1 ASSIGNMENT STATEMENTS

Syntax:

$$\langle \text{assignment-statement} \rangle ::= \langle \text{variable-list} \rangle = \langle \text{formula} \rangle ; \quad (5.0)$$
$$\langle \text{variable-list} \rangle ::= \langle \text{variable} \rangle, \dots \quad (6.1)$$

Semantics:

An <assignment-statement> causes the value of the <formula> to the right of the equal sign to be assigned to the <variables> to the left of the equal sign.

In performing the assignment, the `<formula>` is evaluated first. Then, the leftmost variable is evaluated and the value of the formula is assigned to that variable. Next, the second-to-the-left variable is evaluated and the value of the formula is assigned to it. This sequence of evaluations continues until the list of variables is exhausted. If necessary and permitted (see Section 7.0), the value of the formula is implicitly converted to the type of the variable being assigned to. For numeric values, the value is rounded or truncated according to the `<round-or-truncate>` attribute of each variable being assigned to (see Sections 2.1.1.2 and 2.1.1.3).

Constraints:

The type of the $\langle \text{formula} \rangle$ must match or be implicitly convertible to that of each of the $\langle \text{variables} \rangle$ according to the rules given in Section 7.0.

All <variables> in the <variable-list> must be of the same type class.

None of the <variables> may be <formal-input-parameters>.

Note:

Assignment semantics and constraints apply to presets (Section 2.1.6), assignments to <control-items> in <loop-statements> (Section 4.2), and some types of actual/formal parameter correspondence (Section 3.3).

Since the implemented precision of packed fixed point table items may be less than the implemented precision of an unpacked item having the same fixed type, and since rounding and truncation are performed with respect to implemented precision, assignment to packed table items may change the value being assigned (see Section 7.0).

4.2 LOOP STATEMENTS

Syntax:

<loop-statement>	::= <loop-type> <controlled-statement>	
<loop-type>	::= <while-clause> <for-clause>	
<controlled-statement>	::= <statement>	
<while-clause>	::= WHILE <boolean-formula> ;	(5.2.2)
<for-clause>	::= FOR <control-item> : <control-clause> ;	
<control-item>	::= <control-variable> <control-letter>	
<control-variable>	::= <item-name>	(2.1.1)
<control-letter>	::= <letter>	(8.1)
<control-clause>	::= <initial-value> [<continuation>]	

<initial-value>	::= <formula>	(5.0)
<continuation>	::= <by-or-then phrase> [<while-phrase>] <while-phrase> [<by-or-then-phrase>]	
<by-or-then-phrase>	::= <by-phrase> <then-phrase>	
<by-phrase>	::= BY <by-formula>	
<by-formula>	::= <numeric-formula>	(5.1)
<then-phrase>	::= THEN <formula>	(5.0)
<while-phrase>	::= WHILE <boolean-formula>	(5.2.2)

Semantics:

A <loop-statement> provides for the iterative execution of a statement.

If the <while-clause> form of the <loop-statement> is used, the <controlled-statement> is executed until the value of the <boolean-formula> becomes FALSE. The <boolean-formula> is evaluated before each iteration.

If the <for-clause> form is used, the value of <control-item> determines the number of iterations. If the <control-item> is an <item-name>, its type is as specified in its <declaration>, and that <item-name> may be used for purposes other than loop control before and after the loop. After execution of the loop concludes, the value of the <item-name> is the last value it received in the <loop-statement>. If the <control-item> is a <letter>, the <for-clause> constitutes an implicit declaration of the <control-item>, and its value is inaccessible prior to the start of the <loop-statement> and after the <loop-statement> concludes. Its type is that of the <initial-value>. Its scope is the <loop-statement> itself; hence, another loop statement may use the same <letter> as a <control-item> (except as prohibited in Constraints) and no conflict will result.

The actions of the <loop-statement> with a <for-clause> are as specified by the following algorithm:

Step 1: The <initial-value> is evaluated and assigned to the <control-item>.

Step 2: The <boolean-formula> in the <while-phrase> (if present) is evaluated. If it is FALSE, execution of the <loop-statement> concludes.

Step 3: The <controlled-statement> is executed.

Step 4: The formula in the <by-or-then-phrase> (if present) is evaluated. The value for the <by-formula> (if present) is added to the <control-item>. The value of the <then-formula> (if present) is assigned to the <control-item>. Execution continues at Step 2.

The <control-item> may be used in a <formula> in the <control-clause> and in the <controlled-statement>.

Execution of the <loop-statement> concludes if control is passed to another statement by means of a GOTO, RETURN, EXIT, STOP, or ABORT statement.

Constraints:

If the <control-item> is a <letter>, it must not be used in the <controlled-statement> or <control-clause> in any context in which its value can be altered (e.g., as an <actual-output-parameter> or as a target in an <assignment-statement>). If the <control-item> is an <item-name>, assignments to it in the <controlled-statement> are not prohibited, but will result in a warning message.

A <label> in a <controlled-statement> cannot be used as the <statement-name> in a <goto-statement> or <abort-phrase> that is outside the <controlled-statement> or as an <actual-input-parameter> in a subroutine invocation that is outside the <controlled-statement>.

The <initial-value>, <by-formula> and <then-formula> must match or be implicitly convertible to the type of the <control-item> (see Section 7.0). Further, the sum of the <by-formula> and the <control-item> must match or be implicitly convertible to the type of the <control-item>. The <initial-value> cannot be of type table.

The <by-formula> (if present) must have type and value such that it may be legally added to the <control-item> according to the rules of Sections 5.1.1, 5.1.2, and 5.1.3.

If the <control-item> is a <control-letter>, the <initial-value> must not be a <status-constant> that belongs to more than one type (unless the <status-constant> is disambiguated by an explicit conversion -- see Section 7).

The <control-letter> in a <loop-statement> may not be the same as

the <control-letter> of any enclosing <loop-statement>.

A <bit-formula> cannot be implicitly converted to the <boolean-formula> in a <while-phrase>.

4.3 IF STATEMENTS

Syntax:

<if-statement>	::= IF <boolean-formula> ; <conditional-statement> [<else-clause>]	(5.2.2)
<conditional-statement>	::= <statement>	(4.0)
<else-clause>	::= [<directive>...] ELSE <statement>	(9.0) (4.0)

Semantics:

An <if-statement> provides for conditional execution of a statement depending on the value of its <boolean-formula>.

If the value of the <boolean-formula> is TRUE, the <conditional-statement> is executed and the <statement> in the ELSE clause (if any) is not executed.

If the value of the <boolean-formula> is FALSE, the <statement> in the <else-clause> (if present) is executed rather than the <conditional-statement>. In the event of nested <if-statements>, an ELSE associates with the innermost unmatched IF.

If the <boolean-formula> has a value that is known at compile time, conditional compilation (see Section 1.2.4) will occur.

Constraints:

<Labels> throughout a scope must be unique, even if portions of the text within the scope are unselected as a result of conditional compilation.

<Directives> preceding ELSE must be text directives (Section 9.2) or listing directives (Section 9.7).

A <bit-formula> cannot be implicitly converted to the

<boolean-formula> in an <if-statement>.

Note:

<Labels> in the <conditional-statement> and in the <else-clause> are in the same scope as the <if-statement> itself.

4.4 CASE STATEMENTS

Syntax:

```

<case-statement> ::= CASE
                   <case-selector-formula> ;
                   [<directive>...] (9.0)
                   BEGIN <case-body>
                   [<label>...] END (4.0)

<case-selector-formula> ::= <integer-formula> (5.1.1)
                        | <bit-formula> (5.2)
                        | <character-formula> (5.3)
                        | <status-formula> (5.4)

<case-body> ::= <case-alternative>...

<case-alternative> ::= [ <directive>... ] (9.0)
                     <case-index-group>
                     <statement> (4.0)
                     [FALLTHRU]
                     | <default-option>

<default-option> ::= [<directive>...] (9.0)
                   ( DEFAULT ) ;
                   <statement> (4.0)
                   [FALLTHRU]

<case-index-group> ::= ( <case-index>,... ) :

<case-index> ::= <compile-time-integer-formula> (5.1.1)
               | <compile-time-bit-formula> (5.1.2)
               | <compile-time-character- (5.1.3)
                 formula>

```


- | <compile-time-status-formula> (5.4)
- | <lower-bound> : (2.1.2.1)
- | <upper-bound> (2.1.2.1)

Semantics:

Whereas an <if-statement> provides for the optional execution of either of two statements, a <case-statement> provides for a choice of executing one or more of a number of statements. (The possible choices are represented by the various <case-alternatives>).

The particular <case-alternative> is selected according to the value of <case-selector-formula>. Several values of the <case-selector-formula> may select the same <case-alternative>.

With the exception of the <default-option>, each <case-alternative> is headed by a <case-index-group> that designates the possible values of the <case-selector-formula> that after being implicitly converted (if necessary) to the type of the <case-selector-formula>, cause that particular <case-alternative> to be selected for execution. Each <case-index> can designate either a single value or, for integer and status selector types, a closed range of values bounded by <lower-bound> and <upper-bound>.

If the value of the <case-selector-formula> does not correspond to a <case-index> value, the <statement> in the <default-option> is executed.

If FALLTHRU is not present after a selected <statement>, execution of the <case-statement> concludes after that <statement> is executed. If FALLTHRU is present after the selected <statement>, the <statement> in the textually-succeeding <case-alternative> is then executed. Control continues to "fall through" to subsequent <case-alternatives>, until a case-alternative with no FALLTHRU is executed or until the END of the <case-statement> has been reached.

If the value of the <case-selector-formula> is known at compile time, conditional compilation (see Section 1.2.4) will occur for all unselected alternatives that cannot be reached via FALLTHRU semantics.

Constraints:

No two <case-alternatives> within the same <case-statement> can be associated with identical <case-index> values.

If a <default-option> is not present, the value of the <case-selector-formula> must be represented by a <case-index>.

The types of each formula in a <case-index> must match or be implicitly convertible to that of the <case-selector-formula> according to the rules given in Section 7.0.

If the <case-selector-formula> is a <status-formula>, a <case-index> specifying lower and upper bounds is legal only if the status-type has the default representation (see Section 2.1.1.6).

The <upper-bound> in a <case-index> must be greater than or equal to the <lower-bound>.

<Directives> preceding DEFAULT or <case-index-groups> must be text directives (Section 9.27) or listing directives (Section 9.7).

Within a <case-statement>, at most one <default-option> may be used as a <case-alternative>.

Note:

<Labels> in the <default-option> and in the <case-alternatives> are in the same scope as the <case-statement> itself. Consequently, control can be transferred into or between case statements.

4.5 PROCEDURE CALL STATEMENTS

Syntax:

```

<procedure-call-
statement>          ::= <user-defined-procedure-call>
                        | <machine-specific-procedure-call>
<user-defined-procedure-
call>                ::= <procedure-name>                      (3.1)
                        {<actual-parameter-list>}
                        [<abort-phrase>] ;
<actual-parameter-list> ::= ( [<actual-input-parameter>,...]
                        [ : <actual-output-parameter>,... ] )
<actual-input-parameter> ::= <formula>                        (5.0)
                        | <statement-name>                    (4.0)
                        | <function-name>                     (3.2)
                        | <procedure-name>                    (3.1)

```

	<block-name>	(2.1.4)
	<block-dereference>	(6.1)
	<nested-block>	
<nested-block>	::= <block-name>	(2.1.4)
	[<block-dereference>]	(6.1)
<actual-output-parameter>	::= <variable>	(6.1)
	<block-name>	(2.1.4)
	<block-dereference>	(6.1)
	<nested-block>	
<abort-phrase>	::= ABORT <statement-name>	(4.0)
<machine-specific-procedure-call>	::= <procedure-name>	(3.1)
	[<actual-parameter-list>] ;	

Semantics:

A <procedure-call-statement> causes invocation of a procedure and the association of formal parameters with actual parameters according to the rules given in Section 3.3.

A <user-defined-procedure-call> causes invocation of a procedure defined in a <procedure-definition>. The <abort-phrase> is for use in connection with <abort-statements>. Its semantics are explained in Section 4.10.

A <machine-specific-procedure-call> causes invocation of a machine-specific procedure (see Section 3.5).

A <nested-block> is a block contained in another block. If the <block-name> was declared in a <block-type-declaration>, the <block-dereference> references the particular block from which the nested block is to be obtained.

Constraints:

Actual parameters in the <procedure-call-statement> must match the formal parameters of the called procedure in number, kind, and parameter list position, according to the rules given in Section 3.3.

06 June 1980

The `<statement-name>` in an `<abort-phrase>` or `<actual-input-parameter>` must be known in the scope in which the `<procedure-call-statement>` appears, but it must not name a statement that is in another module or in an enclosing subroutine or that was in unselected text in conditional compilation. It cannot be the name of a statement that is in a `<controlled-statement>` unless the `<procedure-call-statement>` itself is within that same `<controlled-statement>`.

4.6 RETURN STATEMENTS

Syntax:

```
<return-statement> ::= RETURN ;
```

Semantics:

The effect of a `<return-statement>` is to terminate the execution of a subroutine, set any parameters that have value-result semantics, and return control to the point following the invocation of the subroutine. If the `<return-statement>` is in a `<function-body>`, the current value of the `<function-name>` becomes the value of the function call.

If the subroutine containing the `<return-statement>` is nested within any enclosing subroutines, only the innermost subroutine is terminated.

Constraint:

The `<return-statement>` can appear only within the body of a subroutine.

4.7 GOTO STATEMENTS

Syntax:

```
<goto-statement> ::= GOTO <statement-name> ; (4.0)
```

Semantics:

A `<goto-statement>` causes control to be transferred to the statement named by the specified `<statement-name>`.

When the `<statement-name>` is a formal statement-name parameter, the effect of a `<goto-statement>` is equivalent to returning from the current subroutine invocation without setting value-result parameters and then executing a `<goto-statement>` at the point of the subroutine's

invocation.

Constraints:

The <statement-name> must be known in the scope in which the <goto-statement> appears. Further, the <statement-name> must not be the <label> of a statement that is in an enclosing subroutine or in another module. It cannot be the <label> of a statement in a <controlled-statement> unless the <goto-statement> is itself within that same <controlled-statement>.

4.8 EXIT STATEMENTS

Syntax:

<exit-statement> ::= EXIT ;

Semantics:

An <exit-statement> causes execution of the immediately enclosing <loop-statement> to terminate. Its effect is the same as a GOTO statement that transfers control out of the <controlled-statement> to the point following the end of the <loop-statement>.

Constraint:

The <exit-statement> can appear only in a <controlled-statement>.

4.9 STOP STATEMENTS

Syntax:

<stop-statement> ::= STOP [<integer-formula>] ; (5.1.1)

Semantics:

A <stop-statement> causes execution of the <complete-program> to terminate. If a <stop-statement> is executed within a <subroutine-body>, the value-result <actual-output-parameters> of any subroutine whose call is still active will not be set.

The value of the optional <integer-formula> in a <stop-statement> is made available to the environment in which the J73 program is executing, where its semantics are implementation-dependent. Absence of an <integer-formula> implies the value is not determined.

Constraint:

The range of legal values of the <integer-formula> is MINSTOP through MAXSTOP.

4.10 ABORT STATEMENTS

Syntax:

<abort-statement> ::= ABORT ;

Semantics:

When an <abort-statement> is executed, control passes to the statement named in the <abort-phrase> of the most recently executed, currently active <procedure-call-statement> that has an <abort-phrase>. All intervening subroutine invocations are terminated, and value-result parameters of such subroutines are not set. If there is no currently-active <procedure-call-statement> that has an <abort-phrase>, the effect of the <abort-statement> is the same as STOP.

5.0 FORMULAS

Syntax:

<formula> ::= <numeric-formula> (5.1)

! <bit-formula> (5.2)

! <character-formula> (5.3)

! <status-formula> (5.4)

! <pointer-formula> (5.5)

! <table-formula> (5.6)

<compile-time-formula> ::= <compile-time numeric formula> (5.1)

! <compile-time bit formula> (5.2)

! <compile-time character-formula> (5.3)

! <compile-time status formula> (5.4)

! <compile-time pointer formula> (5.5)

Semantics:

<Formulas> represent values. Each <formula> has associated with it a type class and appropriate attributes.

A <compile-time-formula> is a <formula> whose value is computed and used at compile time.

All compile-time computations are performed using the range and precision parameters of the target machine.

The following constructions yield values at compile time.

1. Data declared in <constant-item-declarations>, except for constant items whose type class is pointer.
2. The functions LBOUND, FIRST, and LAST, regardless of their arguments; the function UBOUND, provided its argument is not a table with * dimensions; the functions NEXT, BIT, BYTE, SHIFTL, SHIFTR, ABS, and SGN, provided their arguments are known at compile time; the function NWDSSEN, provided its argument does not contain a

reference to a name whose declaration is not completed prior to the point at which the function appears; the functions BITSIZE, BYTESIZE, and WORDSIZE, provided (1) their arguments are known at compile time, (2) their arguments do not contain references to names whose declarations are not completed prior to the points at which the functions appear, and (3) their arguments are not blocks and are not tables with * dimensions.

3. All operator-operand combinations other than dereferencing, indexing, and assignment, provided the operands have values that are known at compile time.
4. All type conversions except REP, provided the value of the <formula> being converted is known at compile time.
5. All machine parameters.
6. All <status-constants>.
7. All <literals>.

The following values are not known at compile time:

1. Constant items whose type class is pointer.
2. Constant tables and their components.
3. All data declared without the word CONSTANT.
4. The LOC function, regardless of its argument; the function UBOUND, if its argument is a table with * dimensions; the functions NEXT, BIT, BYTE, SEIBL, SHIFTR, ARS, and SEN, if they have one or more arguments whose values are not known at compile time; the function NWDSN, if its argument is a name whose declaration is not completed prior to the point at which the function appears; the functions BITSIZE, BYTESIZE, and WORDSIZE, if (1) their arguments have values that are not known at compile time, (2) their arguments contain references to names whose declarations are not completed prior to the points at which the functions appear, or (3) their arguments are blocks or tables with * dimensions.
5. All operator-operand combinations that have one or more evaluated operands whose values are not known at compile time.

06 June 1980

6. The REP conversion.
7. Any value arrived at via a <statement>.
8. Dereferenced or subscripted values.

Any value known at compile time may also be used as a run-time value.

5.1 NUMERIC FORMULAS

Syntax:

<numeric-formula>	::= <integer-formula>	(5.1.1)
	<floating-formula>	(5.1.2)
	<fixed-formula>	(5.1.3)
<compile-time-numeric-formula>::=	<compile-time-integer-formula>	(5.1.1)
	<compile-time-floating-formula>	(5.1.2)
	<compile-time-fixed-formula>	(5.1.3)

Semantics:

A <numeric-formula> represents a numeric value.

A <compile-time-numeric-formula> represents a numeric value that is known at compile time (see Section 5.0).

5.1.1 INTEGER FORMULAS

Syntax:

<integer-formula>	::= [<sign>] <integer-term>	(8.3.1)
	<integer-formula>	
	<plus-or-minus>	(8.2.3)
	<integer-term>	
<integer-term>	::= <integer-factor>	
	<integer-term>	
	<multiply-divide-or-mod>	(8.2.3)
	<integer-factor>	

<integer-factor>	::= <integer-primary>	
	<integer-factor> **	
	<integer-primary>	
<integer-primary>	::= <integer-literal>	(8.3.1)
	<integer-machine-	
	parameter>	(1.4)
	<integer-variable>	
	<named-integer-constant>	
	<integer-function-call>	
	(<integer-form 'a>)	
	<integer-conversion>	(7.0)
	(<formula>)	(5.0)
<integer-variable>	::= <variable>	(6.1)
<named-integer-constant>	::= <named-constant>	(6.2)
<integer-function-call>	::= <function-call>	(6.3)
<compile-time-integer-formula>	::= <integer-formula>	

Semantics:

An <integer-formula> represents a value whose type class is integer, i.e., S or U.

The integer operators are +, -, *, /, MOD, and **, which denote addition, subtraction, multiplication, division, modulus, and exponentiation, respectively.

The type of a formula composed of an integer operator and two operands is S NN-1, where NN is the actual number of bits that would be supplied by the implementation for a signed integer <item-declaration> whose size attribute is the larger of the size attributes of the two operands. The type of an <integer-formula> consisting of a <sign> and an <integer-term> is S NN-1, where NN is the actual number of bits that would be supplied for a signed integer <item-declaration> whose size attribute is that of the <integer-term>.

The quotient of two integers is first computed exactly and then truncated to an integer result. Truncation will be toward zero.

MIL-STD-1589B (USAF)
06 June 1980

The modulus of two integers, $AA \text{ MOD } BB$, is equivalent to $AA - (AA/BB)*BB$.

the value produced by integer exponentiation to a positive power is the same as that produced by repeated multiplication.

The value produced by integer exponentiation to a negative power is $1 / (\text{base} ** \text{abs}(\text{power}))$ and in most cases is zero.

Constraints:

The value of an $\langle \text{integer-formula} \rangle$ with size attribute SS must lie in the range $\text{MININT}(SS)$ through $\text{MAXINT}(SS)$.

An $\langle \text{integer-variable} \rangle$, $\langle \text{named-integer-constant} \rangle$, or $\langle \text{integer-function-call} \rangle$ must be an integer (S or U) type.

A $\langle \text{compile-time-integer-formula} \rangle$ must be a $\langle \text{integer-formula} \rangle$ whose value is known at compile-time (see Section 5.0).

The right operand of $/$ and MOD must be non-zero.

Note:

R and T used in an explicit conversion (see Section 7.0) do not affect the value of integer division.

5.1.2 FLOATING FORMULAS

Syntax:

$\langle \text{floating-formula} \rangle$	$::=$	$[\langle \text{sign} \rangle] \langle \text{floating-term} \rangle$	(8.3...
		$ \langle \text{floating-formula} \rangle$	
		$\langle \text{plus-or-minus} \rangle$	(8.2.3)
		$\langle \text{floating-term} \rangle$	
$\langle \text{floating-term} \rangle$	$::=$	$\langle \text{floating-factor} \rangle$	
		$ \langle \text{floating-term} \rangle$	
		$\langle \text{multiply-or-divide} \rangle$	(8.2.3)
		$\langle \text{floating-factor} \rangle$	
$\langle \text{floating-factor} \rangle$	$::=$	$\langle \text{floating-primary} \rangle$	
		$ \langle \text{floating-factor} \rangle$	
		$** \langle \text{floating-primary} \rangle$	

	<floating-factor> ** <integer-primary>	(5.1.1)
<floating-primary>	::= <floating-literal>	(8.3.1)
	<floating-machine-parameter>	(1.4)
	<floating-variable>	
	<named-floating-constant>	
	<floating-function-call>	
	(<floating-formula>)	
	<floating-conversion>	(7.0)
	(<formula>)	(5.0)
<floating-variable>	::= <variable>	(6.1)
<named-floating-constant>	::= <named-constant>	(6.2)
<floating-function-call>	::= <function-call>	(6.3)
<compile-time-floating-formula>	::= <floating-formula>	

Semantics:

A <floating-formula> represents a value whose type class is float.

The floating operators are +, -, *, /, and **, which denote addition, subtraction, multiplication, division, and exponentiation respectively. In exponentiation with a <floating-factor>, a floating value is produced in all cases.

The precision attribute of a <floating-formula> is that of the formula's most precise floating operand. The operand of a <floating-conversion> is first computed according to the default rules, and then converted to the specified floating type (see Section 7.0).

For floating exponentiations whose right operand is an <integer-primary>, the result is -(ABS (left operand) ** right operand) if left operand is negative and right operand is odd; (ABS (left operand) ** right operand) in all other cases.

Constraints:

The value of a <floating-formula> with precision PP must lie in the range FLOATUNDERFLOW (II) through MAXFLOAT (II) or the range MINFLOAT (II) through -FLOATUNDERFLOW (II) or be zero, where II = IMPLFLOATPRECISION(PP).

A <floating-variable>, <named-floating-constant>, or <floating-function-call> must be a floating type.

A <compile-time-floating-formula> must be a <floating-formula> whose value is known at compile time (see Section 5.0).

For exponentiations where the right operand is a <floating-primary>, the left operand must not be negative.

Exponentiation of an integer base to a floating power cannot be performed. Either the base must be converted to floating or the power must be converted to integer.

The divisor must be non-zero.

Note:

The round or truncate attribute associated with variables or constant names does not affect the computation of floating formula results. Floating formulas are evaluated in an implementation-dependent manner with respect to how exact results are approximated to the implemented precision.

5.1.3 FIXED FORMULAS

Syntax:

<fixed-formula>	::=	[<sign>] <fixed-term>	(8.3.1)
		<fixed-formula> <plus-or-minus> <fixed-term>	(8.2.3)
<fixed-term>	::=	<fixed-factor>	
		<fixed-term> * <fixed-factor>	
		<integer-term> * <fixed-factor>	(5.1.1)
		<fixed-term> <multiply-or-divide>	(8.2.3)

	<integer-factor>	(5.1.1)
<fixed-factor>	::= <fixed-literal>	(8.3.1)
	<fixed-machine-parameter>	(1.4)
	<fixed-variable>	
	<named-fixed-constant>	
	<fixed-function-call>	
	(<fixed-formula>)	
	<fixed-conversion>	(7.0)
	(<fixed-term> /	
	<fixed-factor>)	
	<fixed-conversion>	(7.0)
	(<integer-term> /	
	<fixed-factor>)	
	<fixed-conversion>	(7.0)
	(<formula>)	(5.0)
<fixed-variable>	::= <variable>	(6.1)
<named-fixed-constant>	::= <named-constant>	(6.2)
<fixed-function-call>	::= <function-call>	(6.3)
<compile-time-fixed-formula>	::= <fixed-formula>	

Semantics:

A <fixed-formula> represents a fixed point value.

The fixed point operators are +, -, *, and /, which denote addition, subtraction, multiplication, and division, respectively. The rules specifying the result type of these operators guarantee that, in general, exact results are produced. The specific rules are given below for each operator. In these rules, S_n, F_n, and P_n refer to the scale, fraction part, and precision of an operand or result and n is 1, 2, or R to indicate the first operand, second operand, or result, respectively.

For addition and subtraction, the default type of the result is:

$$\begin{aligned} SR &= S1 = S2 \\ FR &= \text{Max} (F1, F2) \end{aligned}$$

$$PR = \text{Max } (P1, P2)$$

For multiplication, there are two cases:

1. When one operand is an integer, the result scale and precision is that produced by successive addition, i.e.,

$$\begin{aligned} SR &= Sa \\ PR &= Pa \\ FR &= Fa \end{aligned}$$

where Sa, Fa, and Pa represent the scale, fraction, and precision values of the fixed point operand.

2. When both operands are fixed point types, the type of the result is:

$$\begin{aligned} SR &= S1 + S2 \\ PR &= P1 + P2 \\ FR &= F1 + F2 \end{aligned}$$

If PR is larger than MAXFIXEDPRECISION or if SR does not lie in the range - 127 through + 127, then the product must be explicitly converted to a valid fixed point scale and precision (see Section 7.0).

For division, there are also two cases:

1. When dividing a fixed point value by an integer, the scale and precision of the result are the scale and precision of the numerator. Truncation will be toward zero.
2. When both operands are fixed point values or when an integer is divided by a fixed point value, the result is exact and must be explicitly converted to a programmer specified scale and precision (see Section 7.0).

The default result type of a <fixed-formula> containing a <sign> as a prefix operator is the type of the operand.

The result type of a <fixed-factor> that is a <fixed-variable>, <named-fixed-constant>, or <fixed-function-call> is the type specified in their respective variable, constant, or function declarations.

The type of a <fixed-literal> is contextually determined (see Section 8.3.1).

The result type of a <fixed-formula> enclosed in parentheses is the type of the enclosed <fixed-formula>.

06 June 1980

The result type of a <fixed-factor> containing a <fixed-conversion> is the type specified by the <fixed-conversion>. If the operand of the <fixed-conversion> is a <fixed-term> or <fixed-formula>, the infix or unary operator is evaluated exactly, and the mathematically-defined result is converted to the specified fixed type.

Constraints:

Except for the operand of a <fixed-conversion>, the value of a <fixed-formula> whose scale is SS and whose fraction attribute is FF must lie in the range MINFIXED(SS,PP-SS) through MAXFIXED(SS,PP-SS), where PP = IMPLFIXEDPRECISION (SS,FF).

A <fixed-variable>, <named-fixed-constant>, and <fixed-function-call> must have been declared as fixed types.

Operands of fixed point addition or subtraction must have identical scales.

A <compile-time-fixed-formula> must be a <fixed-formula> whose value is known at compile time (see Section 5.0).

The divisor must be non-zero.

Note:

MOD and ** are not defined for fixed point operands.

5.2 BIT FORMULAS

Syntax:

<bit-formula>	::= <logical-operand> [<logical-continuation>] NOT <logical-operand>	
<logical-operand>	::= <bit-primary> <relational-expression>	(5.2.1)
<bit-primary>	::= <bit-literal> <boolean-literal> <bit-variable>	(8.3.2) (8.3.3)

		<named-bit-constant>	
		<bit-function-call>	
		(<bit-formula>)	
		<bit-conversion>	(7.0)
		(<formula>)	(5.0)
<logical-continuation>	::=	<and-continuation>...	
		<or-continuation>...	
		<xor-continuation>...	
		<eqv-continuation>...	
<and-continuation>	::=	AND <logical-operand>	
<or-continuation>	::=	OR <logical operand>	
<xor-continuation>	::=	XOR <logical-operand>	
<eqv-continuation>	::=	EQV <logical-operand>	
<bit-variable>	::=	<variable>	(6.1)
<named-bit-constant>	::=	<named-constant>	(6.2)
<bit-function-call>	::=	<function-call>	(6.3)
<compile-time-bit-formula>	::=	<bit-formula>	

Semantics:

A <bit-formula> represents a value whose type class is bit. Its size is the number of bits comprising its value.

If the <bit-formula> is composed of <logical-operands> and one or more of the logical operators AND, OR, XOR, and EQV, the size of the result is the size of the longest operand. Shorter operands are padded on the left with zeros as necessary. Note that the syntax requires explicit parentheses for all <bit-formulas> containing two or more of these operators, unless the operators are identical.

NOT produces a value that is the logical complement of its operand. AND, OR (inclusive or), XOR (exclusive or), and EQV (equivalence) perform their usual logical function on their two operands on a bit-by-bit basis. If both operands have a size of one bit and the value

of the left operand is such that the result of the operator can be determined, evaluation is "short-circuited", i.e., the right operand will not be evaluated and need only satisfy semantic constraints that can always, even in the most general case, be verified without evaluating the operand (e.g., the operand need not satisfy the division by zero constraint if it is not evaluated).

Constraints:

A <bit-variable> must be a <variable> whose type class is bit.

A <named-bit-constant> must be a <named-constant> whose type class is bit.

A <bit-function-call> must be a <function-call> whose result value is bit.

A <compile-time-bit-formula> must be a <bit-formula> whose value is known at compile time (see Section 5.0).

5.2.1 RELATIONAL EXPRESSIONS

Syntax:

<relational-expression>	::=	<integer-formula>	(5.1.1)
		<relational-operator>	(8.2.3)
		<integer-formula>	(5.1.1)
		<floating-formula>	(5.1.2)
		<relational-operator>	(8.2.3)
		<floating-formula>	(5.1.2)
		<fixed-formula>	(5.1.3)
		<relational-operator>	(8.2.3)
		<fixed-formula>	(5.1.3)
		<character-formula>	(5.3)
		<relational-operator>	(8.2.3)
		<character-formula>	(5.3)
		<status-formula>	(5.4)
		<relational-operator>	(8.2.3)
		<status-formula>	(5.4)
		<bit-primary>	(5.2)
		<equal-or-not-equal-operator>	(8.2.3)
		<bit-primary>	(5.2)

<pointer-formula>	(5.5)
<relational-operator>	(8.2.3)
<pointer-formula>	(5.5)

Semantics:

A <relational-expression> represents a value obtained by comparing two formulas using a <relational-operator>. Its type class is B and its size is one bit.

The relational operators, = (equal), <> (not equal), < (less than), > (greater than), <= (less than or equal), and >= (greater than or equal), carry their usual meanings.

Character comparisons will be made on the basis of the collating sequence of the character set used in a given implementation.

Status comparisons will be made on the basis of the representation of the status values.

Pointer comparisons will be made on a target-machine-dependent basis.

For bit and character operands, the shorter will be implicitly converted to the type of the longer as described in Section 7.0.

Constraints:

When both operands are <status-constants>, at least one must be unambiguously associated with a single status type.

When the two operands are <status-formulas>, their types must be identical.

When the two operands are <pointer-formulas>, their types must be identical or one must be an untyped pointer.

When both operands are <fixed formulas>, there must exist a type to which both operands are implicitly convertible.

2 5.2.2 BOOLEAN FORMULAS

Syntax:

<boolean-formula> ::= <bit-formula> (5.2)

Semantics:

A <boolean-formula> is <bit-formula> whose size is one bit. It has the value TRUE if the value of the bit is one and FALSE otherwise.

Constraints:

In contexts syntactically requiring a <boolean-formula> (<if-statements>, <while-phrases>, and <trace-controls>), a <bit-formula> cannot be implicitly converted to a <boolean-formula>.

5.3 CHARACTER FORMULAS

Syntax:

<character-formula> ::= <character-literal> (8.3.4)

| <character-variable>

| <named-character-constant>

| <character-function-call>

| (<character-formula>)

| <character-conversion> (7.0)

(<formula>) (5.0)

<character-variable> ::= <variable> (6.1)

<named-character-constant> ::= <named-constant> (6.2)

<character-function-call> ::= <function-call> (6.3)

<compile-time-character-formula> ::= <character-formula>

Semantics:

A <character-formula> represents a value whose type class is character. Its size is the number of bytes comprising its value.

Constraints:

A <character-variable> must be a <variable> whose type class is character.

A <named-character-constant> must be a <named-constant> whose type class is character.

A <character-function-call> must be a <function-call> whose result value is character.

A <compile-time-character-formula> must be a <character-formula> whose value is known at compile time (see Section 5.0).

5.4 STATUS FORMULAS

Syntax:

<status-formula>	::= <status-constant>	(2.1.1.6)
	<status-variable>	
	<named-status-constant>	
	<status-function-call>	
	(<status-formula>)	
	<status-conversion>	(7.0)
	(<formula>)	(5.0)
<status-variable>	::= <variable>	(6.1)
<named-status-constant>	::= <named-constant>	(6.2)
<status-function-call>	::= <function-call>	(6.3)
<compile-time-status-formula>	::= <status-formula>	

Semantics:

A <status-formula> represents a value whose type class is status.

Constraints:

A <status-variable> must be a <variable> whose type class is status.

A <named-status-constant> must be a <named-constant> whose type class is status.

A <status-function-call> must be a <function-call> whose result value is status.

A <compile-time-status-formula> must be a <status-formula> whose value is known at compile time (see Section 5.0).

5.5 POINTER FORMULAS

Syntax:

<pointer-formula>	::= <pointer-literal>	(8.3.5)
	<pointer-variable>	
	<named-pointer-constant>	
	<pointer-function-call>	
	(<pointer-formula>)	
	<pointer-conversion>	(7.0)
	(<formula>)	(5.0)
<pointer-variable>	::= <variable>	(6.1)
<named-pointer-constant>	::= <named-constant>	(6.2)
<pointer-function-call>	::= <function-call>	(6.3)
<compile-time-pointer-formula>	::= <pointer-formula>	

Semantics:

A <pointer-formula> represents a value whose type class is pointer.

Constraints:

A <pointer-variable> must be a <variable> whose type class is pointer.

A <named-pointer-constant> must be a <named-constant> whose type class is pointer.

A <pointer-function-call> must be a <function-call> whose result value is pointer.

MIL-STD-1589B (USAF)
06 June 1980

A <compile-time-pointer-formula> must be a <pointer-formula> whose value is known at compile time (see Section 5.0).

5.6 TABLE FORMULAS

Syntax:

<table-formula>	::= <table-variable>	
	<named-table-constant>	
	(<table-formula>)	
	<table-conversion>	(7.0)
	(<formula>)	(5.0)
<table-variable>	::= <variable>	(6.1)
<named-table-constant>	::= <named-constant>	(6.2)

Semantics:

A <table-formula> represents a value whose type class is table.

Constraints:

A <table-variable> must be a <variable> whose type class is table.

A <named-table-constant> must be a <named-constant> whose type class is table.

6.0 DATA REFERENCES

6.1 VARIABLES

Syntax:

<u><variable></u>	::= <named-variable> <bit-function-variable> <byte-function-variable> <rep-function-variable> <function-name>	(3.2)
<u><named-variable></u>	::= <item> <table> <table-item> <table-entry> <block-item> <block-table> <block-table-item> <block-table-entry>	
<u><item></u>	::= <item-name> <item-dereference>	(2.1.1)
<u><table></u>	::= <table-name> <table-dereference>	(2.1.2)
<u><table-item></u>	::= <table-item-name> [<subscript>] [<table-dereference>]	(2.1.2.3)
<u><table-entry></u>	::= <table-name> <subscript> <table-dereference> <subscript>	(2.1.2)

<block-item>	::= <item-name> [<block-dereference>]	(2.1.1)
<block-table>	::= <table-name> [<block-dereference>]	(2.1.2)
<block-table-item>	::= <table-item-name> [<subscript>] [<block-dereference>]	(2.1.2.3)
<block-table-entry>	::= <table-name> <subscript> [<block-dereference>]	(2.1.2)
<block-dereference>	::= <dereference>	
<item-dereference>	::= <dereference>	
<table-dereference>	::= <dereference>	
<dereference>	::= @ <pointer-item-name> @ (<pointer-formula>)	(5.5)
<pointer-item-name>	::= <item-name> <table-item-name> <constant-item-name>	(2.1.1) (2.1.2.3) (2.1.3)
<subscript>	::= (<index>, ...)	(5.1.1)
<index>	::= <integer-formula> <status-formula>	(5.1.1) (5.4)
<bit-function-variable>	::= BIT (<bit-variable> , <fbit> , <nbit>)	(5.2) (6.3.3)
<byte-function-variable>	::= BYTE (<character-variable> , <fbyte> , <nbyte>)	(5.3) (6.3.4)
<rep-function-variable>	::= <rep-conversion> (<named-variable>)	(7.0)

Semantics:

A <variable> designates a data object whose value can be changed by assignment. A <named-variable> designates a data object whose value can be used in a formula and changed by assignment. A <dereference> designates the data object whose address is contained in the <pointer-item-name> or <pointer-formula> of the <dereference>.

An <item> variable designates either an object declared in an item declaration or an object pointed to by a typed pointer whose type-name attribute is an item type. In the latter case the item is referenced with an <item-dereference> (i.e., the pointer is dereferenced to obtain the item).

A <table> variable designates either an object declared in a table declaration or an object pointed to by a typed pointer whose type-name attribute is a table type. In the latter case the table is referenced with a <table-dereference> (i.e., the pointer is dereferenced to obtain the table). The type class of a <table> is table.

A <table-item> variable designates an item component of a table. If the table is dimensioned, the subscript indicates from which entry the item is to be obtained. If <table-item-name> was declared in a <table-type-declaration> (rather than a <table-item-declaration>) the <table-dereference> references the particular table from which the item is to be obtained.

A <table-entry> variable designates an entry in a dimensioned table. The table is referenced either with a <table-name> or with a <table-dereference>.

The type class of a <table-entry> is table for entries declared with an <ordinary-table-body>, <specified-table-body>, or <table-type-name>, and otherwise is the type specified by the underlying <item-type-description>. (Note that <table-entry> is syntactically a subscripted <table-name> or <table-dereference>.)

If the type class of a particular <table-entry> is not table, any operation or intrinsic function except LOC, NWDSN, and REP applied to that entry is interpreted as applying to the item whose type class and attributes are given by the underlying <item-type-description>. LOC, NWDSN, and REP are interpreted as applying to the entire physical space occupied by the object, including filler bits preceding or following the item.

A <block-item> variable designates an item component of a block. If the <item-name> was declared in a <block-type-declaration>, the <block-dereference> references the particular block from which the item is to be obtained.

A <block-table> variable designates a table component of a block. If the <table-name> was declared in a <block-type-declaration>, the <block-dereference> references the particular block from which the table is to be obtained.

A <block-table-item> variable designates an item component of a table which is itself a component of a block. If the table is dimensioned, the subscript indicates from which entry the item is to be obtained. If the <table-item-name> was declared in a <block-type-declaration>, the <block-dereference> references the particular block from which the item is to be obtained. (Note that if the <table-item-name> was declared in a <table-type-declaration>, it cannot be obtained as a <block-table-item> variable but must be obtained as a <table-item> variable with a <table-dereference>.)

A <block-table-entry> variable designates an entry in a dimensioned table which is contained in a block. If the <table-name> was declared in a <block-type-declaration>, the <block-dereference> references the particular block from which the table entry is to be obtained.

A <bit-function-variable> is the use of the BIT function in an assignment context (i.e., the target of an assignment statement or an actual output parameter) to designate that a specified substring of the <bit-variable> is to be used as a variable. <Fbit> indicates the starting bit and <nbit> indicates the size of the substring. Bits are numbered from the left beginning with zero.

A <byte-function-variable> is the use of the BYTE function in an assignment context (i.e., a target of an assignment statement or an actual output parameter) to designate that a specified substring of the <character-variable> is to be used as a variable. <Fbyte> indicates the starting character and <nbyte> indicates the size of the substring. Characters are numbered from the left beginning with zero.

A <rep-function-variable> is the use of the <rep-conversion> in an assignment context (i.e., the target of an assignment statement or an actual output parameter) to designate that the <named-variable> is to be treated as a bit string variable whose size is the number of bits of storage actually occupied by the <named-variable>.

Constraints:

A <subscript> must be present in a <table-item> or <block-table-item> if the type of the table is dimensioned.

A <subscript> in a <table-item>, <table-entry>, <block-table-item>, or <block-table-entry> must contain the same number of <indices> as there are <dimensions> in the <dimension-list> of the declaration of the

table's type. Furthermore, the type of each <index> must be the same as the type of the corresponding <dimension> and the value of each index must be within the bounds specified for that dimension. If the designated table is a formal parameter and the <dimensions> were specified as *, the indices must be <integer-formulas> (even if bounds of an actual parameter on a particular invocation are of status type), and the value of each index must be in the range 0 through NN-1, where NN is the number of elements in that dimension of the actual parameter.

If the <table-item-name> in a <table-item> was declared in a <table-type-declaration>, the <table-item> must contain a <table-dereference> whose pointer is of the appropriate type.

If the <item-name> in a <block-item> was declared in a <block-type-declaration>, the <block-item> must contain a <block-dereference> whose pointer is of the appropriate type.

A reference to a <table-item> must not access storage outside the bounds of the table containing that <table-item>.

If the <table-name> in a <block-table> or <block-table-entry> was declared in a <block-type-declaration>, the <block-table> or <block-table-entry> must contain a <block-dereference> whose pointer is of the appropriate type.

If the <table-item-name> in a <block-table-item> was declared in a <block-type-declaration>, the <block-table-item> must contain a <block-dereference> whose pointer is of the appropriate type.

<Fbit> and <nbit> must not designate a substring beyond the bounds of the <bit-variable>. <Nbit> must be greater than zero.

<Fbyte> and <nbyte> must not designate a substring beyond the bounds of the <character-variable>. <Nbyte> must be greater than zero.

A <function-name> can be used as a <variable> only within the body of a function having that <function-name>, and then only as the left-hand side of an assignment statement. The other valid uses of <function-name> are described in Section 3.2.

A pointer to an undimensioned parallel or tight table type cannot be used in a <dereference>.

The value of a pointer used in a <dereference> must be in the implementation-defined set of valid values for pointers of its type. A pointer whose value is NULL cannot be dereferenced.

6.2 NAMED CONSTANTS

Syntax:

<code><named-constant></code>	<code>::= <constant-item-name></code>	(2.1.3)
	<code><constant-table-name></code>	(2.1.3)
	<code><constant-table-item-name></code> <code>[<subscript>]</code>	(6.1)
	<code><constant-table-name></code> <code><subscript></code>	(2.1.3) (6.1)
	<code><control-letter></code>	(4.2)
<code><constant-table-item-name></code>	<code>::= <table-item-name></code>	(2.1.2.3)

Semantics:

A `<named-constant>` designates a constant data object whose value can be used in a formula but cannot be changed.

A `<constant-item-name>` designates an object declared in a constant item declaration.

A `<constant-table-name>` designates an object declared in a constant table declaration.

A `<constant-table-item-name>` designates an item component of a constant table. If the table is dimensioned, the `<subscript>` indicates from which entry the item is to be obtained.

A `<constant-table-name>` followed by a `<subscript>` designates an entry in a dimensioned constant table.

A `<control-letter>` designates an object created in a `<for-clause>` whose `<control-item>` is a single letter.

Constraints:

A `<subscript>` must follow a `<constant-table-item-name>` if the table is dimensioned.

A `<subscript>` following a `<constant-table-item-name>` or `<constant-table-name>` must contain the same number of `<indices>` as there are `<dimensions>` in the `<dimension-list>` in the declaration of the table. Furthermore, the type of each `<index>` must be the same as the type of the corresponding `<dimension>` and the value of each `<index>` must be

within the bounds specified for that <dimension>.

Constant tables and items selected from constant tables via subscripts cannot be used as compile-time values.

A <control-letter> may be referenced only within the <controlled-statement> of a <loop-statement> whose <for-clause> created that <control-constant>.

6.3 FUNCTION CALLS

Syntax:

```
<function-call>          ::= <user-defined-function-call>
                           | <intrinsic-function-call>
                           | <machine-specific-function-call>

<user-defined-function-call> ::= <function-name>           (3.2)
                                [<actual-parameter-list>]   (4.5)

<intrinsic-function-call>  ::= <loc-function>              (6.3.1)
                                | <next-function>           (6.3.2)
                                | <bit-function>            (6.3.3)
                                | <byte-function>           (6.3.4)
                                | <shift-function>          (6.3.5)
                                | <abs-function>             (6.3.6)
                                | <sign-function>            (6.3.7)
                                | <size-function>           (6.3.8)
                                | <bounds-function>         (6.3.9)
                                | <nwdsen-function>         (6.3.10)
                                | <status-inverse-function> (6.3.11)

<machine-specific-function-
call>                      ::= <function-name>           (3.2)
                                [<actual-parameter-list>]   (4.5)
```

Semantics:

Execution of a <function-call> causes invocation of a function. Any actual parameters are bound to the corresponding formal parameters as described in Section 3.3.

A <user-defined-function-call> causes invocation of a function defined in a <function-definition>. The type of the value returned by the function is the type specified by the <item-type-description> in the <function-heading> of the <function-definition>.

An <intrinsic-function-call> causes invocation of a language-defined function. A description of the language-defined functions is contained in the following sections. The type of the value returned by each function is described in the corresponding section.

A <machine-specific-function-call> causes invocation of a machine-specific function (see Section 3.5).

Constraints:

Actual parameters in the <function-call> must match the formal parameters of the called function in number, type, and parameter list position according to the rules given in Section 3.3.

6.3.1 LOC FUNCTION

Syntax:

<loc-function>	::= LOC (<loc-argument>)	
<loc-argument>	::= <named-variable>	(6.1)
	<block-name>	(2.1.4)
	<statement-name>	(4.0)
	<procedure-name>	(3.1)
	<function-name>	(3.2)
	<block-dereference>	(6.1)

Semantics:

The LOC function can be applied to the <loc-argument> to obtain the machine address of the word in which the <loc-argument> is stored. If the <loc-argument> is a <named-variable> or <block-name> that was

declared with a <type-name> TT, the type of the value returned by the LOC function is P TT (i.e., a typed pointer). Otherwise, the type of the value returned by the LOC function is P (i.e., an untyped pointer).

If the <loc-argument> is a <statement-name>, <procedure-name>, or <function-name> the <loc-function> yields an untyped pointer whose value is the machine address used to access the designated statement or subroutine.

Constraints:

The LOC of a subroutine whose name appears in an <inline-declaration>, or of a <statement-name> whose definition appears in such a subroutine, is implementation-defined.

Note:

The LOC function cannot be applied to an intrinsic function.

6.3.2 NEXT FUNCTION

Syntax:

<next-function>	::= NEXT (	
	<next-argument> ,	
	<increment-amount>)	
<next-argument>	::= <pointer-formula>	(5.5)
	<status-formula>	(5.4)
<increment-amount>	::= <integer-formula>	(5.1.1)

Semantics:

If the <next-argument> is a <pointer-formula>, the value returned by the NEXT function is the arithmetic sum of the representation of the <pointer-formula> plus the <increment-amount> * LOCSINWORD (i.e., the <pointer-formula> is treated as an integer). The type of the value returned is a pointer of the same type as the <next-argument>.

If the <next-argument> is a <status-formula> and the value of the <increment-amount> is N, the value returned by the NEXT function is the Nth successor (or predecessor if N is negative) of the value of the <status-formula> in this <status-list>. The type of the value is the same as the type of the <next-argument>.

Constraints:

The <next-argument> cannot be a <status-constant> that belongs to more than one status type (unless explicitly disambiguated with a <status-conversion>), nor can it be the <pointer-literal> NULL.

The type of the <status-formula> must be a status type with a default representation.

When the <next-argument> is a <status-formula>, the <increment-amount> must not cause the NEXT function to return a value out of range of the type of the <next-argument>.

The value of the <pointer-formula> and the value of the pointer result must be in the implementation-defined set of valid values for pointers of its type.

Note:

The value of the <next-argument> may be negative.

6.3.3 BIT FUNCTION

Syntax:

<bit-function>	::= BIT (<bit-formula> , <fbit> , <nbit>)	(5.2)
<fbit>	::= <integer-formula>	(5.1.1)
<nbit>	::= <integer-formula>	(5.1.1)

Semantics:

The BIT function selects a designated substring from the <bit-formula>. <Fbit> indicates the starting bit and <nbit> indicates the size of the substring. Bits are numbered from the left beginning with zero. The type of the value returned is a bit string of the same size as the <bit-formula>. The designated substring is right justified in the result and padded on the left with zero bits as necessary to fill the size.

Constraints:

<Fbit> and <nbit> must not designate a substring beyond the bounds of the <bit-formula>. <Nbit> must be greater than zero.

6.3.4 BYTE FUNCTION

Syntax:

<byte-function>	::= BYTE (<character-formula> , <fbyte> , <nbyte>)	(5.3)
<fbyte>	::= <integer-formula>	(5.1.1)
<nbyte>	::= <integer-formula>	(5.1.1)

Semantics:

The **BYTE** function selects a designated substring from the **<character-formula>**. **<Fbyte>** indicates the starting character and **<nbyte>** indicates the size of the substring. Characters are numbered from the left beginning with zero. The type of the value returned is a character string of the same size as the **<character-formula>**. The designated substring is left justified in the result, and padded on the right with blanks as necessary to fill the size.

Constraints:

<Fbyte> and <nbyte> must not designate a substring beyond the bounds of the <character-formula>. <Nbyte> must be greater than zero.

6.3.5 SHIFT FUNCTIONS

Syntax:

$$\begin{aligned} \langle \text{shift-function} \rangle &::= \langle \text{shift-direction} \rangle \\ &\quad (\langle \text{bit-formula} \rangle , \\ &\quad \langle \text{shift-count} \rangle) \quad (5.2) \\ \langle \text{shift-direction} \rangle &::= \text{SHIFTL} \\ &\quad | \text{SHIFTR} \\ \langle \text{shift-count} \rangle &::= \langle \text{integer-formula} \rangle \quad (5.1.1) \end{aligned}$$

Semantics:

The SHIFTL function performs a logical left shift of the <bit-formula> by the number of positions indicated by <shift-count>. The SHIFTR function performs a logical right shift of the <bit-formula> by the number of positions indicated by <shift-count>. In both cases, vacated bits are filled with zeros and bits shifted out are lost. If the <shift-count> is greater than or equal to the size of the

<bit-formula>, the result is a bit string with all zero bits. The type of the value returned by a <shift-function> is the same as the type of the <bit-formula>.

Constraints:

The value of <shift-count> must be non-negative and less than or equal to MAXBITS.

6.3.6 ABS FUNCTION

Syntax:

<abs-function> ::= ABS (<numeric-formula>) (5.1)

Semantics:

The ABS function produces a value that is the absolute value of the <numeric-formula>. The result is equivalent to - <numeric-formula> if <numeric-formula> is negative and equivalent to + <numeric-formula> otherwise.

6.3.7 SIGN FUNCTION

Syntax:

<sign-function> ::= SGN (<numeric-formula>) (5.1)

Semantics:

The SGN function returns a value according to the following rules:

<u>Numeric Formula</u>	<u>Value</u>
> 0	+1
= 0	0
< 0	-1

The type of the value is S 1.

6.3.8 SIZE FUNCTIONS

Syntax:

```

<size-function>      ::= <size-type>
                        ( <size-argument> )

<size-type>          ::= BITSIZE
                        | BYTESIZE
                        | WORDSIZE

<size-argument>      ::= <formula>                      (5.0)
                        | <block-name>                  (2.1.4)
                        | <type-name>                   (2.1.1.7)

```

Semantics:

The BITSIZE, BYTESIZE and WORDSIZE functions return the logical size of the <size-argument> in bits, bytes, and words respectively. The type of the value returned is S MAXINTSIZE. The logical BITSIZE of each data type in the language will be described below. The logical BYTESIZE is equal to BITSIZE/BITSINBYTE if BITSIZE MOD BITSINBYTE = 0 and BITSIZE/BITSINBYTE+1 otherwise. Similarly, the logical WORDSIZE is equal to BITSIZE/BITSINWORD if BITSIZE MOD BITSINWORD = 0 and BITSIZE/BITSINWORD+1 otherwise.

Bit: The BITSIZE of an object of type B NN is NN

Integer: The BITSIZE of an object of type U NN is NN and S NN is NN+1

Fixed: The BITSIZE of an object of type A MM, NN is MM+NN+1

Float: The BITSIZE of a float object is the number of bits of storage the object actually occupies.

Character: The BITSIZE of an object of type C NN is NN*BITSINBYTE.

Pointer: The BITSIZE of a pointer object is BITSINPOINTER.

Status: The BITSIZE of a status object is the <status-size>. If <status-size> was specified, the BITSIZE is specified in the object's <status-item-description>. If no <status-size>

was specified, the BITSIZE is minimum number of bits of storage needed to represent objects of that type.

Table: The BITSIZE of a table or table entry that is not tightly structured is the number of bits from the leftmost bit of the first word occupied by the table or table entry to the rightmost bit of the last word occupied by the table or table entry. The BITSIZE of a tightly structured table entry is <bits-per-entry>. The BITSIZE of a tightly structured table is the number of bits from the leftmost bit of the first word occupied by the table to the rightmost bit of the last entry, where the last entry occupies <bits-per-entry> bits. Note: the BITSIZE of a <table-entry> whose type class is not table is the BITSIZE of the item specified by the underlying <item-type-description>.

Block: The BITSIZE of a block is $NN * BITSINWORD$, where NN is the number of words the block occupies.

Constraints:

A BITSIZE function must not be applied to a table whose size in words exceeds $MAXINT(MAXINTSIZE)/BITSINWORD$.

A BYTESIZE function must not be applied to a table whose size in words exceeds $MAXINT(MAXINTSIZE)/BYTESINWORD$.

6.3.9 BOUNDS FUNCTIONS

Syntax:

<bounds-function> ::= <which-bound>
(<table-name> , (2.1.2)
<dimension-number>)

<which-bound> ::= LBOUND
| UBOUND

<dimension-number> ::= <compile-time-integer-formula> (5.1.1)

Semantics:

The LBOUND function returns the lower bound of the specified dimension of the designated table. The UBOUND function returns the upper bound of the specified dimension of the designated table. A <dimension-number> of zero refers to the leftmost <dimension> in that table's <dimension-list>; a <dimension-number> of one designates the

next-to-leftmost <dimension> in the list, etc. The type of the returned value will either be an integer type or a status type depending on the declaration of the designated table. If the table is a formal parameter with a * dimension, the type will always be integer, LBOUND will always return zero, and UBOUND will return NN-1, where NN is the number of elements in that dimension of the actual parameter.

Constraints:

The <dimension-number> must be greater than or equal to 0 and less than the number of dimensions in the designated table.

6.3.10 NWDSSEN FUNCTION

Syntax:

```
<nwdsen-function>      ::= NWDSSEN ( <nwdsen-argument> )  
  
<nwdsen-argument>      ::= <table-name>                (2.1.2)  
                        | <table-type-name>              (2.2)
```

Semantics:

The NWDSSEN function returns the number of words of storage allocated to each entry in the named table or table type. The return type is S with default size.

6.3.11 STATUS INVERSE FUNCTIONS

Syntax:

```
<status-inverse-function> ::= FIRST (   
                                <status-inverse-argument> )  
                                | LAST (   
                                <status-inverse-argument> )  
  
<status-inverse-argument> ::= <status-formula>          (5.4)  
                                | <status-type-name>      (2.1.1.6)
```

Semantics:

The FIRST function gives the value of the lowest-valued <status-constant> in the <status-list> associated with the <status-inverse-argument>. The LAST function gives the value of the

MIL-STD-1589B (USAF)
06 June 1980

highest-valued <status-constant> in the <status-list> associated with the <status-inverse-argument>.

The return value has the type indicated by the <status-inverse-argument>.

7.0 TYPE MATCHING AND TYPE CONVERSIONS

Syntax:

```
<bit-conversion>      ::= <bit-type-conversion>
                        | <rep-conversion>

<bit-type-conversion> ::= (* <bit-type-description> *)      (2.1.1.4)
                        | <bit-type-name>                    (2.1.1.4)
                        | B

<integer-conversion>  ::= (* <integer-type-description> *)  (2.1.1.1)
                        | <integer-type-name>                (2.1.1.1)
                        | S
                        | U

<floating-conversion> ::= (* <floating-type-description> *) (2.1.1.2)
                        | <floating-type-name>                (2.1.1.2)
                        | F

<fixed-conversion>    ::= (* <fixed-type-description> *)    (2.1.1.3)
                        | <fixed-type-name>                    (2.1.1.3)

<character-conversion> ::= (* <character-type-description> *) (2.1.1.5)
                        | <character-type-name>                (2.1.1.5)
                        | C

<status-conversion>   ::= (* <status-type-name> *)           (2.1.1.6)
                        | <status-type-name>                    (2.1.1.6)

<pointer-conversion>  ::= (* <pointer-type-description> *)  (2.1.1.7)
                        | <pointer-type-name>                  (2.1.1.7)
                        | P
```


<table-conversion> ::= (* <table-type-name> *) (2.1.2)
 | <table-type-name> (2.1.2)

<rep-conversion> ::= REP

Semantics:

In Section 2.1, the definition of type was given. In some cases, implicit conversions will be performed to achieve type equivalence. In this section, for each type class, rules will be given regarding when two types are the same, when an object of one type will be implicitly converted to another type, and when and how an object of one type can be explicitly converted to another type. Implicit conversions will never be performed on arguments to explicit conversions or when the types of the data objects are required to match exactly. With all the conversions (both implicit and explicit), if the value produced after conversion is not in the range of values of the type being converted to, the conversion is illegal.

For purposes of type equivalence, a user-defined <type-name> is considered an abbreviation for its specification.

A <formula> may be explicitly converted to another type by enclosing it in parentheses and preceding it with appropriate conversion. Note that if the conversion does not consist of a single letter or name, it must be enclosed in (* and *).

Omitted attribute specifiers in type conversions imply the same default values as for declarations of those types.

Type equivalence and conversion rules for each of the J73 type classes are as follows:

Bit (B)

Type Equivalence:	Two bit types are equivalent if their size attributes are equal.
Implicit Conversions:	A bit string will be implicitly converted to a bit string with a different size attribute, with truncation on the left or padding with zeros on the left. Implicit truncation is not permitted when the syntax requires a <boolean-formula>.
Explicit Conversions:	Any data object except a block may be explicitly converted to a bit string with a <bit-conversion>. A <bit-conversion> may be

06 June 1980

either a <bit-type-conversion> or a <rep-conversion>.

A <bit-type-conversion> to a type B NN takes the rightmost NN bits of the data object's representation. If there are fewer than NN bits, the object will be padded on the left with zeroes. The default value for NN is 1. A <bit-type-conversion> may be applied to a data object of any type. If the object being converted is a table or table entry, all "filler" bits (i.e., bits that contribute to the size of the table but that are not part of the component objects' sizes as declared) are included in the string. If the object to be converted is of type class character, filler bits between bytes and unused bytes following the end of the string are not included.

A <rep-conversion> provides a means of obtaining the representation of a data object. A <rep-conversion> treats a data object as a bit string whose size is the number of bits actually occupied by the object. This includes all filler bits and the bits in the unused (but allocated) bytes following the ends of character strings. For all objects whose type class is table, the number of bits in the bit string is the same as the BITSIZE of the object. For all <table-entries> whose type class is not table, the number of bits in the bit string is the total number of bits (including filler bits) in the table entry. A <rep-conversion> can appear in the target of an assignment statement (see Section 6.1). A <rep-conversion> can be applied to <named-variables> only; further, it cannot be applied to tables declared with * dimensions, to entries in parallel tables, or to tables whose size in bits exceeds MAXBITS.

Integer (S and U)

Type Equivalence:

Two integer types are equivalent if they are both S or U and if their size attributes are equal.

Implicit Conversions:

An integer type will be implicitly converted to any other integer type.

Explicit Conversions: An <integer-conversion> is used to explicitly convert a data object to an integer type. The conversion can be applied to objects of bit, integer, fixed, float, and pointer only.

A bit string will be treated as representing the value of the integer type if the size of the bit string is less than or equal to the BITSIZE of the integer type. Otherwise, the conversion is illegal. If the size of the bit string is less than the BITSIZE of the integer type, the bitstring will be padded on the left with zeroes.

An integer, fixed, or floating data object will be converted to the integer type, with truncation or rounding if specified.

Converting a pointer to an integer type is equivalent to first converting the pointer to type B BITSINPOINTER and then converting the bit string to integer.

Floating (F)

Type Equivalence: Two floating types are equivalent if their precision attributes are equal.

Implicit Conversions: A floating type will be implicitly converted to a floating type of the same or greater precision regardless of the round-or-truncate attribute. A <real-literal> will be implicitly treated as a <floating-literal> in the contexts specified in Section 8.3.1. (Implicit floating conversions do not change numeric values although they may cause a change in how the value is represented.)

Explicit Conversions: A <floating-conversion> is used to explicitly convert a data object to a floating data type. The conversion can be applied to <real-literals> and to objects of bit, integer, fixed, and float types only.

A bit string will be treated as representing the value of the floating type if the size of the bit string equals the BITSIZE of the floating type. Otherwise the conversion is illegal.

06 June 1980

An integer, fixed, or floating data object will be converted to the floating type, with truncation or rounding as specified in the <floating-conversion>. Rounding and truncation are performed with respect to the implemented precision of the type specified by the <floating-conversion>.

Fixed (A)

Type Equivalence: Two fixed point types are equivalent if their scale attributes are equal and their fraction attributes are equal.

Implicit Conversions: A fixed point type will be implicitly converted to another fixed point type if the scale and fraction attributes of the target type are both at least as large as those of the source type. A <real-literal> will be implicitly treated as a <fixed-literal> in the contexts specified in Section 8.3.1. Implicit fixed conversions do not change the numeric value represented except when the implemented precision of the result value is less than the implemented precision of the value being converted (see Section 2.1.1.3); in this case, rounding or truncation occurs with respect to the implemented precision of the converted value. This situation occurs only when assigning to a packed fixed table item (in an assignment statement, loop <control-variable>, table preset, or output parameter); the <round-or-truncate> attribute of the table item determines whether the assigned value is rounded or truncated.

Explicit Conversions: A <fixed-conversion> is used to explicitly convert a data object to a fixed point data type. The conversion can be applied to <real-literal> and to objects of bit, integer, fixed, and float types only.

A bit string will be treated as representing the value of the specified fixed point type if the size of the bit string equals the BITSIZE of the fixed point type. Otherwise, the conversion is illegal.

An integer, fixed, or floating data object will be converted to the specified fixed point type, with truncation or rounding as specified in the <fixed-conversion>. Rounding and truncation are performed with respect to the implemented precision of the type specified by the <fixed-conversion>.

Character (C)

Type Equivalence: Two character types are equivalent if their size attributes are equal.

Implicit Conversions: A character string will be implicitly converted to a string with a different size attribute, with truncation on the right or padding with blanks on the right.

Explicit Conversions: A <character-conversion> is used to explicitly convert a data object to a character data type. The conversion can be applied to objects of type bit or character only.

A bit string will be treated as representing the value (excluding filler bits between bytes) of the character type if the size of the bit string equals the BITSIZE of the character type. Otherwise the conversion is illegal.

A character string will be converted to type C NN by taking the leftmost NN characters. If there are fewer than NN characters, the value is padded on the right with blanks.

Pointer (P)

Type Equivalence: Two pointer types are equivalent if they are both untyped pointers or if they are both typed pointers referring to the same <type-declaration>.

Implicit Conversions: A typed pointer will be implicitly converted to an untyped pointer.

Explicit Conversions: A <pointer-conversion> is used to explicitly convert a data object to a pointer type. The conversion can be applied to bit, integer, or pointer data objects only.

A bit string will be treated as representing the value of the pointer type if the size of the bit string equals the BITSIZE of the pointer type. Otherwise the conversion is illegal.

Converting an integer to a pointer is equivalent to first converting the integer to type B BITSINPOINTER and then converting the bit string to a pointer.

Converting a pointer to a different pointer type means that the pointer will be considered as a pointer of the specified type.

Status

Type Equivalence:

Two status types are equivalent if (1) they both have default representation, their size attributes are the same, and both <status-lists> contain the same <status-constants> in the same order, or (2) they both have identical programmer-specified representations, their size attributes are the same, and both <status-lists> contain the same <status-constants>.

Implicit Conversions:

A status type will be implicitly converted to a status type that differs only in its size attribute. Furthermore, a status constant belonging to more than one status type is implicitly disambiguated in the following contexts: (1) when it is the source value of an assignment statement, it takes the type of the target variable; (2) when it is an actual parameter, it takes the type of the corresponding formal parameter; (3) when it is in a table <subscript> or <preset-index-specifier>, it takes the type of the corresponding <dimension> in that table's declaration; (4) when it is a loop <initial-value>, it takes the type of the <control-variable>; (5) when it is in an <item-preset> or <table-preset>, it takes the type of the item or table item being initialized; (6) when it is an operand of a <relational-operator>, it takes the type of the other operand; (7) when it is in a <case-index-group>, it takes the type of the

<case-selector-formula>; and (8) when it is a <lower-bound> or <upper-bound>, it takes the type of the other bound.

Explicit Conversions: A <status-conversion> is used to explicitly convert a data object to a status type. The conversion can be applied to bit or status data objects only.

A bit string will be treated as representing the representational value of the status type if the size of the bit string equals the BITSIZE of the status type and the value of the bit string is within the range of values of the status type. Otherwise the conversion is illegal.

A <status-conversion> may be used to assert the type of a status object. This will be required when a status constant belongs to more than one type and it is used in a context other than these enumerated above under implicit conversions. Except for status objects whose types differ only in their size attributes, a status object cannot be converted to a different status type without first converting it to a bit string.

Table

Type Equivalence:

Two tables have equivalent types if they are both ordinary or both specified, their <structure-specifier> attribute is the same, they have the same number of dimensions, they have the same number of elements in each dimension, they have the same number of items in the same textual order in each entry, the types (including attributes) of the items are equivalent, the (explicit or implied) packing specifier on each of the items is the same (for ordinary tables), the !ORDER directive is either present in both tables or absent in both tables, the <words-per-entry> attribute is the same (for specified tables), and the location-specifiers of the items are the same (for specified tables). (Note that the names of the items, as well as the types and bounds of the dimensions, need not be the same.) A table entry is considered to have no

dimensions. A table whose entry contains an item-declaration is not considered equivalent in type to a table whose entry is declared using an unnamed item description.

Implicit Conversions: No implicit conversions are performed.

Explicit Conversions: A bit or table data object may be explicitly converted to a table type with a <table-conversion>.

A bit string will be treated as representing the value of the table type if the size of the bit string equals the BITSIZE of the table type. Otherwise the conversion is illegal.

A <table-conversion> may be applied to a table object of that type merely to assert its type. (A table object cannot be converted to a different table type without first converting it to a bit string).

8.0 BASIC ELEMENTS

8.1 CHARACTERS

Syntax:

<code><character></code>	<code>::= <letter></code>
	<code> <digit></code>
	<code> <mark></code>
	<code> <other-character></code>
<code><letter></code>	<code>::= A B C D E F</code>
	<code> G H I J K L</code>
	<code> M N O P Q R</code>
	<code> S T U V W X</code>
	<code> Y Z</code>
<code><digit></code>	<code>::= 0 1 2 3 4 5</code>
	<code> 6 7 8 9</code>
<code><mark></code>	<code>::= + - * / > <</code>
	<code> = @ . : , ;</code>
	<code> () ' " % !</code>
	<code> \$ blank</code>

Semantics:

The text of a J73 <complete-program> is a continuous stream of <characters>. However, in some contexts, the end of an input record has significance (see Section 8.2).

Note that in the standard character set for the language <letters> are defined to be upper case letters only. <Marks> are used either alone or in conjunction with other characters as operators, delimiters, and separators. <Other-characters> are the remaining implementation-dependent characters, which are accepted within <character-literals> and <comments>, and which may also be used as described below. Each

implementation must define these characters, as well as the ordering of all <characters> in a collating sequence.

Some of the standard characters are not universally available; therefore, the following standard alternates are defined:

<u>Standard Character</u>	<u>Alternates</u>
@	† or ?
'	-> or _
"	≠
!	V
%	=
:	%

If any of the above standard characters are unavailable on a particular machine, one of the recommended alternates for that character must be used. (The first column of alternates is intended for the CDC standard 63 and 64 character sets; the alternates ? and _ are intended for the Univac 1108.) If the : is replaced, the % must also be replaced.

An implementation that has lower case letters available in addition to uppercase may permit their use in programs provided that within <names>, <reserved-words>, <letters>, <status-constants> and all <literals> except <character-literals> they are considered interchangeable with their corresponding uppercase letters (e.g., XX and xx denote the same name); whereas within <character-literals> they are considered distinct.

An implementation that has square brackets available may allow [to be used for (* and] to be used for *) but may not prohibit the use of the (* and *).

Constraints:

If a left bracket is substituted for (*, then a right bracket must be substituted for the corresponding *). If a right bracket is substituted for *), then a left bracket must be substituted for the corresponding (*.

MIL-STD-1589B (USAF)
06 June 1980

8.2 SYMBOLS

Syntax:

<symbol>	::= <name>	(8.2.1)
	<reserved-word>	(8.2.2)
	<operator>	(8.2.3)
	<literal>	(8.3)
	<status-constant>	(2.1.1.6)
	<comment>	(8.4)
	<define-string>	(2.4)
	<define-call>	(2.4.1)
	<letter>	(8.1)
	<separator>	(8.2.4)

Semantics:

<Characters> are combined into <symbols> to form the vocabulary of the language. <Symbols> are indivisible units and cannot contain blanks, except as noted in Section 8.5. Only <comments>, <define-strings>, define parameters enclosed in quotation marks, <bit-literals>, and <character-literals> may extend across multiple input records; all other symbols are terminated by the end of an input record.

8.2.1 NAMES

Syntax:

<name>	::= <letter-or-\$> <letter-digit-\$-or-prime>...	
<letter-or-\$>	::= <letter> \$	
<letter-digit-\$-or-prime>	::= <letter>	(8.1)

| <digit> (8.1)
| \$
|

Semantics:

<Names> are words having programmer-supplied spellings. <Names> are used to denote entities in the <complete-program>.

Only the first 31 characters of a J73 <name> are used to determine uniqueness. Additional characters are permitted, but are ignored.

For external names, an implementation may further restrict the number of initial characters that determine uniqueness.

A dollar sign in a <name> is translated to an implementation-dependent representation. This translation of the dollar sign permits the use of a character in a <name> that might otherwise be unrepresentable in the language. If, for example, external names in a given system were prefixed by the character '.' a J73 implementation on that system might choose to represent '\$' when it occurs in a name by the representation for '.'. Thus, the name '\$\$ABC' occurring in a source program would be translated '..ABC'.

8.2.2 RESERVED WORDS

Syntax:

<reserved-word> ::= ABORT | AbS | AND | BEGIN | BIT
| BITSIZE | BLOCK | BY | BYREF
| BYRES | BYTE | BYTESIZE | BYVAL
| CASE | COMPOOL | CONDITION*
| CONSTANT | DEF | DEFAULT | DEFINE
| ELSE | ENCAPSULATION* | END | EQV
| EXIT | EXPORTS* | FALLTHRU | FALSE
| FIRST | FOR | FREE* | GOTO
| HANDLER* | IF | IN* | INLINE

MIL-STD-1589B (USAF)
06 June 1980

| INSTANCE | INTERRUPT* | ITEM
| LABEL | LAST | LBOUND | LIKE
| LOC | MOD | NENT* | NEW*
| NEXT | NOT | NULL |
| NWDSN | OR | OVERLAY | PARALLEL
| POS | PROC | PROGRAM | PROTECTED*
| READONLY* | REC | REF | REGISTER*
| RENT | REP | RETURN | SGN
| SHIFTL | SHIFTR | SIGNAL*
| START | STATIC | STATUS | STOP
| TABLE | TERM | THEN | TO*
| TRUE | TYPE | UBOUND | UPDATE*
| WHILE | WITH* | WORDSIZE
| WRITEONLY* | XOR | ZONE*

Semantics:

<Reserved-words> have language-defined meanings and cannot be used as <names>.

Those reserved words followed by an * in the above list are reserved in order to maintain upward compatibility with future extensions to the language and currently have no meaning in J73.

8.2.3 OPERATORS

Syntax:

<operator> ::= <arithmetic-operator>
| <bit-operator>
| <relational-operator>

	<dereference-operator>
	<assignment-operator>
<arithmetic-operator>	::= <plus-or-minus>
	<multiply-divide-or-mod>
	<multiply-or-divide>
	**
<plus-or-minus>	::= +
<multiply-divide-or-mod>	::= * / MOD
<multiply-or-divide>	::= * /
<bit-operator>	::= <logical-operator>
	NOT
<logical-operator>	::= AND OR XOR EQV
<relational-operator>	::= <equal-or-not-equal-operator>
	< > <= >=
<equal-or-not-equal-operator>	::= = <>
<dereference-operator>	::= @
<assignment-operator>	::= =

Semantics:

The meanings of these operators are given in Sections 4, 5, and 6. The order of combination of operators and operands is determined by parentheses and by the operators' precedence. The operation implied by an operator at one precedence level is combined before the operation implied by an operator at a lower level. Within a particular precedence level, operations are combined from left to right if the !LEFTRIGHT directive is in effect and in an implementation-dependent order if the !REARRANGE directive is in effect.

Precedence of operators is defined by the syntax of the language and is summarized below:

MIL-STD-1589B (USAF)
06 June 1980

6 @, subscripting, function calls
5 **
4 *, /, MOD
3 +, -
2 =, <>, <, >, <=, >=
1 NOT, AND, OR, EQV, XOR
0 assignment

8.2.4 SEPARATORS

Syntax:

<separator> ::= (|) | (* | *)
| : | , | ; | !

Semantics:

<Separators> are used for the following purposes in J73:

()	Expression grouping, list delimiters, status constants, position brackets, subscripts, case labels
(* *)	Type conversions
:	Statement name, case label, and preset index terminator; loop control separator; overlay, dimension, subrange, and parameter separator
,	List separator
;	Statement, declaration, and directive terminator
!	Directive indicator, formal define parameter marker

8.3 LITERALS

Syntax:

<literal>	::= <numeric-literal>	(8.3.1)
	<bit-literal>	(8.3.2)
	<boolean-literal>	(8.3.3)
	<character-literal>	(8.3.4)
	<pointer-literal>	(8.3.5)

Semantics:

<Literals> are data objects whose value and type are inherent in the form of the <symbol> itself. Their values are known at compile time, and, like other compile-time values, cannot be altered during execution.

8.3.1 NUMERIC LITERALS

Syntax:

<numeric-literal>	::= <integer-literal>	
	<floating-literal>	
	<fixed-literal>	
<integer-literal>	::= <number>	
<number>	::= <digit>...	(8.1)
<floating-literal>	::= <real-literal>	
<real-literal>	::= <digit>... <exponent>	(8.1)
	<fractional-form>	
	[<exponent>]	
<exponent>	::= E [<sign>] <number>	
<sign>	::= + -	
<fractional-form>	::= <digit>...	(8.1)

| [<digit>...] . <digit>... (8.1)

<fixed-literal> ::= <real-literal>

Semantics:

An <integer-literal>, LL, denotes a decimal value. Its type is S NN, where NN is IMPLINTSIZE(MINSIZE(LL)).

The type of a <real-literal> or a <real-literal> preceded by a <sign> is determined by the context in which the literal appears, namely:

- . when the literal is used as a preset value, it is implicitly converted to the type of the object being preset;
- . when the literal is used as an assignment value, it is implicitly converted to the type of the target being assigned a value;
- . when the literal is an operand of an infix relational or numeric operator and the other operand is not a real-literal, it is converted to the type of the other operand;
- . when the literal is an actual parameter, it is converted to the type of the formal parameter;
- . when a literal is the <initial-value> of a loop <control-clause>, it is converted to the type of the <control-variable>;
- . when the literal is the argument of an explicit fixed or floating conversion, it is converted to the specified type.

If the type of an optionally signed <real-literal> is not determined contextually, it is considered to be a floating type with default precision.

A <real-literal> denotes a decimal value. If an <exponent> is present, the decimal value preceding the <exponent> is multiplied by 10 to the value specified in the <exponent>.

For <real-literals>, non-<exponent> digits in excess of MAXSIGDIGITS will be treated as zeroes in computing the fixed or floating value to be represented.

Contextual determination of the type of a real-literal will not be affected by the presence or absence of the <rearrange-directive>.

Constraints:

<Real-literals> may be implicitly converted to fixed or floating values only.

The value of an <integer-literal> with size SS must not exceed MAXINT(SS).

The value of a <floating-literal> with precision PP must not exceed MAXFLOAT(PP).

The value of a <fixed-literal> with scale SS and fraction FF must not exceed MAXFIXED(SS,FF).

Examples:

ITEM FF F 24 = -0.1;	"equivalent to presetting with (*F 24*) (-0.1)"
ITEM RR F,R 24 = -0.1;	"-0.1 is rounded to a 24 bit mantissa"
ITEM TT F,T 24 = -0.1;	"-0.1 is truncated toward minus infinity"
CONSTANT ITEM CC F,R 24 = 2.5;	
ITEM JJ F,R 24 = CC + .3;	".3 is converted to CC's type"
IF RR > .3; ...	".3 is rounded to a 24 bit mantissa"

Note that if II is an integer it m, then II = 2.5 is illegal, since a <real-literal> cannot be implicitly converted to an integer value.

8.3.2 BIT LITERALS

Syntax:

<bit-literal>	::= <bead-size> B ' <bead>...
<bead-size>	::= 1 2 3 4 5
<bead>	::= <digit>
	A B C D E F
	G H I J K L
	M N O P Q R

MIL-STD-1589B (USAF)
06 June 1980

I S I T I U I V

Semantics:

A <bit-literal> represents a bit string value. A <bit-literal> is composed of a string of <beads> whose <bead-size> in bits is indicated in the specification of the literal. The total size of the <bit-literal> is the <bead-size> times the number of beads enclosed within the primes.

The <beads> of a <bit-literal> can be specified as one to five bits in size. The <digit> preceding the B indicates the <bead-size>. Only those <beads> whose value will fit in the <bead-size> indicated are permitted. The digits 0 - 9 represent their actual values; the letters A - V represent the values 10 - 31 (see Table 8-1).

Table 8-1. Bit-Literal Bead Values

Bead	Minimum Bead Size	Binary Value	Bead	Minimum Bead Size	Binary Value
0	1	0	G	5	10000
1	1	1	H	5	10001
2	2	10	I	5	10010
3	2	11	J	5	10011
4	3	100	K	5	10100
5	3	101	L	5	10101
6	3	110	M	5	10110
7	3	111	N	5	10111
8	4	1000	O	5	11000
9	4	1001	P	5	11001
A	4	1010	Q	5	11010
B	4	1011	R	5	11011
C	4	1100	S	5	11100
D	4	1101	T	5	11101
E	4	1110	U	5	11110
F	4	1111	V	5	11111

MIL-STD-1589B (USAF)
06 June 1980

8.3.3 BOOLEAN LITERALS

Syntax:

<boolean-literal> ::= TRUE
| FALSE

Semantics:

<Boolean-literals> represent the two possible truth values. TRUE is equivalent to 1B'1', and FALSE is equivalent to 1B'0'.

8.3.4 CHARACTER LITERALS

Syntax:

<character-literal> ::= <character>... ' (8.1)

Semantics:

<Character-literals> denote strings of character values.

<Character-literals> can contain any <character> (including blank) that is representable in an implementation. A prime character (') is represented within a <character-literal> by two consecutive primes. The size of a <character-literal> in bytes is the number of characters represented within the containing primes (two consecutive primes represent one character). The encoding of characters is implementation-dependent.

8.3.5 POINTER LITERAL

Syntax:

<pointer-literal> ::= NULL

Semantics:

Any pointer item, regardless of its attribute, can have the value NULL, which indicates that the item points to no object.

8.4 COMMENTS

Syntax:

<comment> ::= " [<character>...] " (8.1)

| % [<character>...] % (8.1)

Semantics:

A <comment> has no semantic effect.

A <comment> in a <define-string> or <actual-define-parameter> is interpreted as part of the character sequence to be substituted when the <define-call> is expanded.

A <comment> can appear between any two <symbols>, subject to the constraints below.

Constraints:

A <comment> delimited by a quotation mark (") is not permitted between a <define-name> and a <define-string> in a <define-declaration>, or within the <actual-parameter-list> in a <define-call>.

A <comment> delimited by a quotation mark cannot contain a quotation mark, and a <comment> delimited by a percent character (%) cannot contain a percent character.

8.5 BLANKS

One or more blanks can be placed between <symbols>. Blanks occurring between <symbols> have no semantic meaning.

Constraints:

Blanks cannot appear within <symbols> except in <character-literals>, <define-strings>, <define-calls>, and <comments>.

One or more blanks must appear between any two <symbols> if the absence of blanks could cause them to be interpreted as a single legal <symbol>, except that whether (*) represents one or two <symbols> is contextually determined, e.g., (*) represents two symbols in the following contexts:

```
TABLE AA (*) ...;  
ITEM ... POS (*, 0);
```

AD-A100 577

AERONAUTICAL SYSTEMS DIV WRIGHT-PATTERSON AFB OH

F/G 1/3

AFSC STANDARDIZATION CONFERENCE, 1553, 1589, 1750, 1760, ADA, N--ETC(U)

NOV 80 E C GANGL, S E SMITH

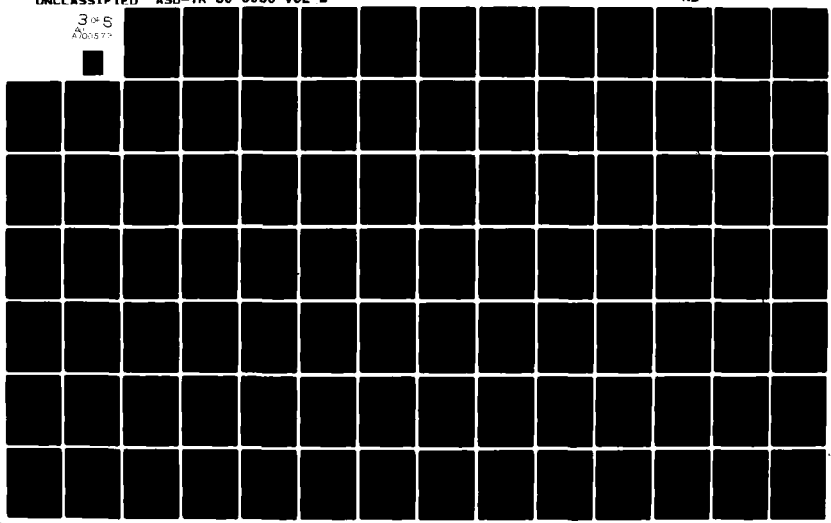
UNCLASSIFIED

ASD-TR-80-5050-VOL-2

NL

3 of 5

AL 700577



9.0 DIRECTIVES

Syntax:

<directive>	::= <compool-directive>	(9.1)
	<copy-directive>	(9.2.1)
	<skip-directive>	(9.2.2)
	<begin-directive>	(9.2.2)
	<end-directive>	(9.2.2)
	<linkage-directive>	(9.3)
	<trace-directive>	(9.4)
	<interference-directive>	(9.5)
	<reducible-directive>	(9.6)
	<nolist-directive>	(9.7.1)
	<list-directive>	(9.7.1)
	<eject-directive>	(9.7.1)
	<listinv-directive>	(9.7.2)
	<listexp-directive>	(9.7.2)
	<listboth-directive>	(9.7.2)
	<base-directive>	(9.8)
	<isbase-directive>	(9.8)
	<drop-directive>	(9.8)
	<leftright-directive>	(9.9)
	<rearrange-directive>	(9.9)
	<initialize-directive>	(9.10)
	<order-directive>	(9.11)

Semantics:

<Directives> are used to provide supplemental information to a compiler about the <complete-program>, and to provide compiler control.

Each implementation can specify <directives> in addition to those described here, but each must conform to the general form for a <directive>. <Directives> begin with an exclamation point and terminate with a semicolon, and the word following the exclamation point must not duplicate that of any language-defined directive.

9.1 COMPOOL DIRECTIVES

Syntax:

```
<compool-directive>      ::= !COMPOOL
                           [<compool-directive-list>] ;

<compool-directive-list> ::= [<compool-file-name>]
                           <compool-declared-name>, ...
                           | ( [<compool-file-name>] )

<compool-declared-name> ::= <name>                                (8.2.1)
                           | ( <name> )                          (8.2.1)

<compool-file-name>      ::= <character-literal>                 (8.3.4)
```

Semantics:

A <compool-directive> is used to access definitions in a compool module.

A <compool-file-name> is an implementation-dependent file name that specifies the desired compool. If it is omitted, an implicit unnamed compool is assumed. A <compool-file-name> enclosed in parentheses implies that all <names> in the compool are to be made available. (This does not include <names> used in the compool that were obtained from other compools.)

If the <compool-directive> contains a list of <compool-declared-names>, only those names (except as noted below) will be made available.

If a <compool-declared-name> is the name of an item, table, or block declared with a <type-name>, that <type-name> is also made available if it is declared in that compool. (For pointer items, this includes the name of the pointed-to-type). If a

`<compool-declared-name>` is a `<table-item-name>`, the name of the table in which it is contained is also made available. If a table name is made available, any `<status-lists>` and `<status-type-names>` associated with its `<dimensions>` are also made available, provided they are declared in the designated compool.

If a <compool-declared-name> is the name of a table or block and is parenthesized, all names declared in the table or block will be made available, as well as all type names referenced in the table or block, provided they are declared in the designated compool. If a <compool-declared-name> is a <table-type-name> or <block-type-name>, the names of these components will be made available whether or not the name is parenthesized.

If a status item name is made available, its associated <status-list> and <status-type-name> (if any) will also be made available, if they were declared in the designated compool.

If a <compool-declared-name> is the name of a subroutine, any <type-names> associated with that subroutine's formal parameters and return value will also be made available, if they are declared in the designated compool.

Constraints:

A <compool-directive> must only occur immediately after START or immediately following another <compool-directive>.

The `<compool-declared-names>` must have been declared in the designated compool.

A <compool-declared-name> cannot be the name of a component declared in a type declaration, nor can it be the name of a formal parameter of a subroutine.

9.2 TEXT DIRECTIVES

9.2.1 COPY DIRECTIVES

Syntax:

```

<copy-directive> ::= !COPY
                    <character-literal> ; (8.3.4)

```

Semantics:

The <copy-directive> is used to copy the contents of a text file into a program. The <copy-directive> can be viewed as a <define-call>; it is expanded at the point of its occurrence by substituting the entirety of the file being copied. The <character-literal> is an implementation-dependent file name.

9.2.2 SKIP, BEGIN, AND END DIRECTIVES

Syntax:

<skip-directive> ::= !SKIP [<letter>] ; (8.1)

<begin-directive> ::= !BEGIN [<letter>] ; (8.1)

<end-directive> ::= !END ;

Semantics:

The <skip-directive> is used in conjunction with a <begin-directive> and an <end-directive> to cause text enclosed in the latter two to be ignored in the process of compilation.

A <skip-directive> with a <letter> will suppress the processing of all text following a <begin-directive> containing the same <letter> up to the matching <end-directive>. A <skip-directive> with no <letter> refers to all <begin-directives>. The text following a <begin-directive> with no <letter> can be suppressed only by a <skip-directive> with no <letter>.

Begin-end directive pairs can be nested. Within a begin-end directive set whose text is being suppressed, enclosed <begin-directives> are recognized for the purpose of matching <end-directives>.

Within a begin-end directive pair whose text is being suppressed, <copy-directives> and <define-calls> will not be expanded.

9.3 LINKAGE DIRECTIVES

Syntax:

<linkage-directive> ::= !LINKAGE
 <symbol>... ; (8.2)

Semantics:

The <linkage-directive> indicates that the specified subroutine does not obey standard J73 linkage conventions. The <symbol> string specifies the implementation-dependent linkage type to be used in linking the procedure.

Constraints:

The <linkage-directive> must only occur in a <subroutine-declaration> or <subroutine-definition> between the heading and the <declarations> of the formal parameters.

If a <subroutine-definition> contains a <linkage-directive>, every <subroutine-declaration> for that subroutine must contain the same <linkage-directive>.

9.4 TRACE DIRECTIVES

Syntax:

```
<trace-directive>      ::= !TRACE  
                        [<trace-control>]  
                        <name>,... ;  
  
<trace-control>        ::= ( <boolean-formula> ) (5.2.2)
```

Semantics:

The <trace-directive> provides a run-time facility to trace program flow and to monitor data assignment. This "tracing" will be active from the lexical point at which the <trace-directive> occurs in the source until the end of the scope containing the directive. Its effect extends into nested procedures declared within this lexical range of statements.

The <names> in the <trace-directive> are the names that will be traced, i.e., certain uses of these names as described in the following sentences will be noted in an implementation-dependent manner, for example on a symbolic output file. For statement names, tracing of the associated statement will be noted each time the statement is fallen into or branched to. For data names, modification of the data object and its new value will be noted. Modification of a data object is considered to have occurred upon execution of an assignment statement in which the data object is the target or upon return from a subroutine to which the data object was passed as an actual output parameter. For tables, modification of the entire table, a table entry, or an item in the table will be noted. For blocks, modification of any data contained in the block will be noted. For subroutine names, each call to the

subroutine will be noted. If the subroutine containing the <trace-directive> is named in the directive and the directive is placed immediately after the <procedure heading> or <function heading>, entry and exit to that subroutine will be noted.

If a <trace-control> appears in the <trace-directive>, the <trace-control> formula will be tested dynamically at each use of a <name> as defined in the preceding paragraph. The trace output is suppressed if the formula is determined to be false. If the <trace-control> is omitted, it is considered to be true.

If two or more active <trace-directives> contain the same <name>, then the lexically latest one overrides the earlier ones for that <name>.

Constraints:

All <names> in the <trace-directive>, including names used in the <trace-control>, except for statement names and subroutine names, must have been declared prior to their use in the <trace-directive>.

A <trace-directive> can occur only within a <statement>.

A <bit-formula> cannot be implicitly converted to the <boolean-formula> in a <trace-control>.

9.5 INTERFERENCE DIRECTIVES

Syntax:

```
<interference-directive>      ::= !INTERFERENCE  
                                <interference-control> ;  
  
<interference-control>        ::= <data-name> :           (2.6)  
                                <data-name>,...           (2.6)
```

Semantics:

The <interference-directive> informs the compiler that it cannot assume that the storage associated with the name to the left of the colon is distinct from the storage associated with the names to the right of the colon. In the absence of an <interference-directive> the compiler can make optimizations on the assumption that distinct <data-names> refer to distinct storage locations. If two <data-names> refer to the same storage location, these optimizations could result in erroneous code. If two <data-names> share the same storage, an assignment to one name should affect the value of the other. If the compiler optimizes on the assumption of non-interference, these

MIL-STD-1589B (USAF)
06 June 1980

semantics might not be preserved.

The compiler is aware of storage overlapping as a result of <specified-table-items> and as a result of the arrangement of data within a single overlay. This overlapping need not be reported via an <interference-directive>. Other instances of overlap, e.g., as a result of absolute addresses in separate overlays, must be stated by the use of an <interference-directive>.

An <interference-directive> can occur only in a <declaration>.

All <data-names> in the <interference-control> must have been declared prior to their use in the <interference-directive>.

9.6 REDUCIBLE DIRECTIVES

Syntax:

<reducible-directive> ::= !REDUCIBLE ;

Semantics:

The <reducible-directive> is used to allow additional optimization of function-calls. A reducible function is one for which all calls with identically-valued actual parameters result in identical function values and output parameter values, and which does not modify any data except actual output parameters and automatic data declared within its own body. If a <reducible-directive> is used to designate such functions as reducible, the compiler may detect the existence of such common calls, save the values returned from the initial call for use in place of any subsequent calls, and delete these subsequent calls.

Constraints:

The <reducible-directive>, if present, must be placed immediately following the semicolon of the <function-heading>.

If a function designated as reducible is both declared and defined, the <reducible-directive> must appear in both the definition of the function and in all declarations of it.

9.7 LISTING DIRECTIVES

9.7.1 SOURCE-LISTING DIRECTIVES

Syntax:

<nolist-directive> ::= !NOLIST ;
<list-directive> ::= !LIST ;
<eject-directive> ::= !EJECT ;

Semantics:

Listing directives are used to provide source listing control information to the compiler. The <nolist-directive> causes suppression of the source listing beginning with the next source line, up to and including the next <list-directive>, which causes the listing to be resumed.

The <eject-directive> causes a page eject of the source listing before listing the following source lines. The <eject-directive> is ignored if the source listing is suppressed.

9.7.2 DEFINE-LISTING DIRECTIVES

Syntax:

<listinv-directive> ::= !LISTINV ;
<listexp-directive> ::= !LISTEXP ;
<listboth-directive> ::= !LISTBOTH ;

Semantics:

Define-listing directives allow programmer control over the text to be included in the source program listing for <define-calls>.

The text contained in the listing for a particular <define-call> depends on the define-listing directive which was in effect at the point of the corresponding <define-declaration> (not on the directive in effect at the point of the <define-call>). If this directive was !LISTINV, then the listing contains the text of the <define-call>; if the directive was !LISTEXP, then the listing contains the expanded string (the <define-string> after substitution of <actual-define-parameters>); if the directive was !LISTBOTH, then the listing contains both the invocation and the expansion.

Each define-listing directive is in effect from the lexical point at which it appears to the end of the current scope or to the point at which the next define-listing directive appears, whichever is first. The default define-listing directive in effect at the beginning of every module is !LISTINV.

Constraint:

<listinv-directives>, <listexp-directives> and
<listboth-directives> may appear only in a <declaration>.

Note:

The effect of a define-listing directive for a particular <define-call> is independent of whether a <nolist-directive> is suppressing the source listing at the point of the <define-declaration> being invoked.

9.8 REGISTER DIRECTIVES

Syntax:

<base-directive>	::= !BASE <data-name>	(2.6)
	<integer-literal> ;	(8.3.1)
<isbase-directive>	::= !ISBASE <data-name>	(2.6)
	<integer-literal> ;	(8.3.1)
<drop-directive>	::= !DROP	
	<integer-literal> ;	(8.3.1)

Semantics:

Register directives affect target-machine register allocation. Each of these three directives uses an <integer-literal> in a target-machine-dependent way to specify which register is affected.

The <base-directive> loads the specified register with the address of the object corresponding to the <data-name>.

The <isbase-directive> directs the compiler to assume that the specified register contains the address of the data object corresponding to the <data-name>, but to take no action to guarantee it.

The <drop-directive> frees the specified register for other use by the compiler in generating code for subsequent statements. Both !BASE and !ISBASE cause the compiler to dedicate the register to the value it currently contains until !DROP or the end of the current scope is encountered.

Register directives may be ignored in implementations for machines on which register allocation is not meaningful.

9.9 EXPRESSION EVALUATION ORDER DIRECTIVES

Syntax:

<leftright-directive> ::= !LEFTRIGHT ;
<rearrange-directive> ::= !REARRANGE ;

Semantics:

If a <leftright-directive> is in effect, operators at the same precedence level are evaluated in left-to-right order within a given <formula>, consistent with the order imposed by parentheses.

If a <rearrange-directive> is in effect, order of evaluation is still constrained by parentheses and operator precedence, but the compiler is otherwise free to rearrange the expression for more optimal code generation, such as by applying associative and commutative laws.

The effect of each of these directives extends from the point at which it appears to the end of the current namespace or to the point at which a different expression-evaluation-order directive appears, whichever is first. At the beginning of each module, a <rearrange-directive> is in effect by default.

9.10 INITIALIZATION DIRECTIVES

Syntax:

<initialize-directive> ::= !INITIALIZE ;

Semantics:

The <initialize-directive> causes all STATIC data objects that are not explicitly initialized via an <item-preset>, <table-preset>, or <block-preset>, to be preset by default to all zero bits.

Its effect extends from the point at which it appears to the end of the current namespace.

Constraint:

The <initialize-directive> may appear only in <declarations>, but not in a <table-body> nor in a <block-body-part> nor in a <subroutine-declaration>.

9.11 ALLOCATION ORDER DIRECTIVES

Syntax:

<order-directive> ::= !ORDER ;

Semantics:

The <order-directive> directs a compiler to allocate storage for the data objects in a block or ordinary table in the order in which their declarations appear in the text of the <block-body-part> or the <ordinary-table-options>. Lexically declared data objects that occur earlier in text are allocated physically lower addresses, and if data objects share a word, lexically earlier data are allocated to the left of later data. In the absence of an <order-directive>, a compiler is free to rearrange the physical storage layout for ease of access or more optimal utilization of memory.

The effect of the <order-directive> extends from the point at which it appears to the end of the current block or table. If the <order-directive> is in a block, its effect extends to the components of any blocks or ordinary tables contained in the block.

If an <order-directive> appears in an <ordinary-table-options> in a <table-type-declaration>, the ordering extends to all tables declared of that type.

Constraints:

A block affected by an <order-directive> cannot contain an <overlay-declaration>.

The <order-directive>, if present, must be the first of the <block-body-options> in the <block-body-part>, or the first of the <ordinary-table-options> in the <ordinary-table-body>.

APPENDIX

CROSS-REFERENCE INDEX

This appendix provides a cross-reference for terminal and non-terminal constructs in the J73 syntax used in this manual. For each construct, columns give the section in the manual where it is defined and the sections where it is used or referenced.

<u>Construct</u>	<u>Definition</u>	<u>References</u>
A		2.1.1.3, 7.0, 8.1, 8.3.2
ABORT		4.5, 4.10, 8.2.2
abort-phrase	4.5	4.5
abort-statement	4.10	4.0
ABS		6.3.6, 8.2.2
abs-function	6.3.6	6.3
absolute-address	2.6	2.6
actual-define-parameter	2.4.1	2.4.1
actual-define-parameter-list	2.4.1	2.4.1
actual-input-parameter	4.5	4.5
actual-output-parameter	4.5	4.5
actual-parameter-list	4.5	4.5, 6.3
allocation-specifier	2.1.5	2.1.1, 2.1.2, 2.1.4
AND		5.2.1, 8.2.2, 8.2.3
and-continuation	5.2	5.2
arithmetic-operator	8.2.3	8.2.3
assignment-operator	8.2.3	8.2.3
assignment-statement	4.1	4.0

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
B		2.1.1.4, 7.0, 8.1, 8.3.2
BASE		9.8
base-directive	9.8	9.0
bead	8.3.2	8.3.2
bead-size	8.3.2	8.3.2
BEGIN		1.2.3, 2.0, 2.1.2.3, 2.1.2.4, 2.1.4, 2.5.1, 2.5.2, 2.7, 3.1, 4.0, 4.4, 8.2.2, 9.2.2
begin-directive	9.2.2	9.0
BIT		6.1, 8.2.2
bit-conversion	7.0	5.2
bit-formula	5.2	4.4, 5.0, 5.2, 5.2.2, 6.3.3, 6.3.5
bit-function	6.3.3	6.3
bit-function-call	5.2	5.2
bit-function-variable	6.1	6.1
bit-item-description	2.1.1.4	2.1.1.4
bit-literal	8.3.2	5.2, 8.3
bit-operator	8.2.3	8.2.3
bit-primary	5.2	5.2, 5.2.1
BITSINBYTE		1.4

<u>Construct</u>	<u>Definition</u>	<u>References</u>
BITSINPOINTER		1.4
BITSINWORD		1.4
BITSIZE		6.3.8, 8.2.2
bit-size	2.1.1.4	2.1.1.4
bits-per-entry	2.1.2.2	2.1.2.2
bit-type-conversion	7.0	7.0
bit-type-description	2.1.1.4	2.1.1, 7.0
bit-type-name	2.1.1.4	2.1.1.4, 7.0
bit-variable	5.2	5.2, 6.1
BLOCK		2.1.4, 2.2, 2.5.1, 8.2.2
block-body-options	2.1.4	2.1.4
block-body-part	2.1.4	2.1.4, 2.2
block-declaration	2.1.4	2.1
block-dereference	6.1	4.5, 6.1, 6.3.1
block-item	6.1	6.1
block-name	2.1.4	2.1.4, 2.5.1, 2.6, 5, 6.3.1, 6.3.8
block-preset	2.1.6	2.1.4
block-preset-list	2.1.6	2.1.6
block-preset-values-option	2.1.6	2.1.6
block-table	6.1	6.1
block-table-entry	6.1	6.1
block-table-item	6.1	6.1

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
block-type-declaration	2.2	2.2
block-type-name	2.2	2.1.1.7, 2.1.4, 2.2
boolean-formula	5.2.2	4.2, 4.3, 9.4
boolean-literal	8.3.3	5.2, 8.3
bounds-function	6.3.9	6.3
BY		4.2, 8.2.2
by-formula	4.2	4.2
by-or-then-phrase	4.2	4.2
by-phrase	4.2	4.2
BYREF		3.3, 8.2.2
BYRES		3.3, 8.2.2
BYTE		6.1, 6.3.4, 8.2.2
byte-function	6.3.4	6.3
byte-function-variable	6.1	6.1
BYTEPOS		1.4
BYTESINWORD		1.4
BYTESIZE		6.3.8, 8.2.2
BYVAL		3.3, 8.2.2
C		2.1.1.5, 7.0, 8.1, 8.3.2
CASE		4.4, 8.2.2
case-alternative	4.4	4.4

<u>Construct</u>	<u>Definition</u>	<u>References</u>
case-body	4.4	4.4
case-index	4.4	4.4
case-index-group	4.4	4.4
case-selector-formula	4.4	4.4
case-statement	4.4	4.0
character	8.1	2.4, 2.4.1, 8.3.4, 8.4
character-conversion	7.0	5.3
character-formula	5.3	4.4, 5.0, 5.2.1, 5.3, 6.3.4
character-function-call	5.3	5.3
character-item-description	2.1.1.5	2.1.1.5
character-literal	8.3.4	8.3, 9.1, 9.2.1
character-size	2.1.1.5	2.1.1.5
character-type-description	2.1.1.5	2.1.1, 7.0
character-type-name	2.1.1.5	2.1.1.5, 7.0
character-variable	5.3	5.3, 6.1
comment	8.4	8.2
compile-time-bit-formula	5.1.2	4.4, 5.0
compile-time-character-formula	5.1.3	4.4, 5.0
compile-time-fixed-formula	5.1.3	5.1
compile-time-floating-formula	5.1.2	1.4, 5.1
compile-time-formula	5.0	2.1.6
compile-time-integer-formula	5.1.1	1.4, 2.1.1.1, 2.1.1.2,

<u>Construct</u>	<u>Definition</u>	<u>References</u>
		2.1.1.3, 2.1.1.4, 2.1.1.5, 2.1.1.6, 2.1.2.1, 2.1.2.2, 2.1.2.4, 2.1.6, 2.6, 4.4, 5.1, 6.3.9
compile-time-numeric-formula	5.1	5.0
compile-time-pointer-formula	5.5	5.0
compile-time-status-formula	5.4	2.1.2.1, 2.1.6, 4.4, 5.0
complete-program	1.1	
compound-def	2.5.1	2.5.1
compound-ref	2.5.2	2.5.2
compound-statement	4.0	4.0
COMPOOL		1.2.1, 8.2.2, 9.1
compool-declaration	2.0	1.2.1, 2.0
compool-declared-name	9.1	9.1
compool-directive	9.1	9.0
compool-directive-list	9.1	9.1
compool-file-name	9.1	9.1
compool-module	1.2.1	1.1
compool-name	1.2.1	1.2.1
CONDITION		8.2.2
conditional-statement	4.3	4.3
CONSTANT		2.1.3, 8.2.2
constant-declaration	2.1.3	2.0, 2.1

<u>Construct</u>	<u>Definition</u>	<u>References</u>
constant-index	2.1.6	2.1.6
constant-item-name	2.1.3	2.1.3, 6.1, 6.2
constant-table-item-name	6.2	6.2
constant-table-name	2.1.3	2.1.3, 6.2
continuation	4.2	4.2
control-clause	4.2	4.2
control-item	4.2	4.2
control-letter	4.2	4.2, 6.2
controlled-statement	4.2	4.2
control-variable	4.2	4.2
COPY		9.2.1
copy-directive	9.2.1	9.0
D		2.1.2.3, 8.1, 8.3.2
data-declaration	2.1	2.0, 2.1.4, 2.5.1, 2.5.2
data-name	2.6	2.6, 3.3, 9.5, 9.8
declaration	2.0	1.2.2, 1.2.3, 2.0, 3.1, 3.2
DEF		1.2.2, 2.5.1, 8.2.2
DEFAULT		4.4, 8.2.2
default-option	4.4	4.4
default-preset-sublist	2.1.6	2.1.6
default-sublist	2.1.1.6	2.1.1.6

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
def-block-instantiation	2.5.1	2.5.1
DEFINE		2.4, 8.2.2
define-call	2.4.1	8.2
define-name	2.4	2.4, 2.4.1
define-string	2.4	2.4, 8.2
definition-part	2.4	2.4
define-declaration	2.4	2.0
def-specification	2.5.1	2.5
def-specification-choice	2.5.1	2.5.1
dereference	6.1	6.1
digit	8.1	8.1, 8.2.1, 8.3.1, 8.3.2
dereference-operator	8.2.3	8.2.3
dimension	2.1.2.1	2.1.2.1
dimension-list	2.1.2.1	2.1.2, 2.1.3, 2.2
dimension-number	6.3.9	6.3.9
directive	9.0	1.2.1, 1.2.2, 1.2.3, 2.0, 2.1.2.3, 2.1.2.4, 2.1.4, 2.5.1, 2.5.2, 3.0, 3.1, 3.2, 4.0, 4.2, 4.4
DROP		9.8
drop-directive	9.8	9.0
E		8.1, 8.3.2

<u>Construct</u>	<u>Definition</u>	<u>References</u>
EJECT	9.7.1	9.7
eject-directive	9.7	9.0
ELSE		4.3, 8.2.2
else-clause	4.3	4.3
ENCAPSULATION		8.2.2
END		1.2.3, 2.0, 2.1.2.3, 2.1.2.4, 2.1.4, 2.5.1, 2.5.2, 2.7, 3.1, 4.0, 4.4, 8.2.2, 9.2.2
end-directive	9.2.2	9.0
entry-size	2.1.2.4	2.1.2.4
entry-specifier	2.1.1	2.1.2, 2.2
equal-or-not-equal-operator	8.2.3	5.2.1, 8.2.3
EQV		5.2, 8.2.2, 8.2.3
eqv-continuation	5.2	5.2
EXIT		4.8, 8.2.2
exit-statement	4.8	4.0
exponent	8.3.1	8.3.1
EXPORTS		8.2.2
external-declaration	2.5	2.0
F		2.1.1.2, 7.0, 8.1, 8.3.2
FALLTHRU		4.4, 8.2.2

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
FALSE		8.2.2, 8.3.3
fbit	6.3.3	6.1, 6.3.3
fbyte	6.3.4	6.1, 6.3.4
FIRST		6.3.11, 8.2.2
fixed-conversion	7.0	5.1.3
fixed-factor	5.1.3	5.1.3
fixed-formula	5.1.3	5.1, 5.1.3, 5.2.2
fixed-function-call	5.1.3	5.1.3
fixed-item-description	2.1.1.3	2.1.1.3
fixed-literal	8.3.1	5.1.3, 8.3.1
fixed-machine-parameter	1.4	5.1.3
FIXEDPRECISION		1.4
fixed-term	5.1.3	5.1.3
fixed-type-description	2.1.1.3	2.1.1, 7.0
fixed-type-name	2.1.1.3	2.1.1.3, 7.0
fixed-variable	5.1.3	5.1.3
floating-conversion	5.1.2	5.1.2
floating-factor	5.1.2	5.1.2
floating-formula	5.1.2	5.1, 5.1.2, 5.2.1
floating-function-call	5.1.2	5.1.2
floating-item-description	2.1.1.2	2.1.1.2
floating-literal	8.3.1	5.1.2, 8.3.1
floating-machine-parameter	1.4	5.1.2

<u>Construct</u>	<u>Definition</u>	<u>References</u>
floating-primary	5.1.2	5.1.2
floating-term	5.1.2	5.1.2
floating-type-description	2.1.1.2	2.1.1, 7.0
floating-type-name	2.1.1.2	2.1.1.2, 7.0
floating-variable	5.1.2	5.1.2
FLOATPRECISION		1.4
FLOATRADIX		1.4
FLOATRELPRECISION		1.4
FLOATUNDERFLOW		1.4
FOR		4.2, 8.2.2
for-clause	4.2	4.2
formal-define-parameter	2.4	2.4
formal-define-parameter-list	2.4	2.4
formal-input-parameter	3.3	3.3
formal-output-parameter	3.3	3.3
formal-parameter-list	3.3	3.1, 3.2
formula	5.0	4.1, 4.2, 4.5, 5.1.1, 5.1.1, 5.1.2, 5.1.3, 5.2, 5.3, 5.4, 5.5, 5.6, 6.3.8
fractional-form	8.3.1	8.3.1
fraction-specifier	2.1.1.3	1.4, 2.1.1.3
FREE		8.2.2
function-body	3.2	3.2

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
function-call	6.3	5.1.1, 5.1.2, 5.1.3, 5.2, 5.3, 5.4
function-declaration	3.2	3.0
function-definition	3.2	3.2
function-heading	3.2	3.2
function-name	3.2	3.2, 3.3, 4.5, 6.1, 6.3, 6.3.1
G		8.1, 8.3.2
GOTO		4.7, 8.2.2
goto-statement	4.7	4.0
H		8.1, 8.3.2
HANDLER		8.2.2
I		8.1, 8.3.2
IF		4.3, 8.2.2
if-statement	4.3	4.0
IMPLFIXEDPRECISION		1.4
IMPLFLOATPRECISION		1.4
IMPLINTSIZE		1.4
IN		8.2.2
increment-amount	6.3.2	6.3.2

<u>Construct</u>	<u>Definition</u>	<u>References</u>
index	6.1	6.1
INITIALIZE		9.10
initialize-directive	9.10	9.0
initial-value	4.2	4.2
INLINE		3.4, 8.2.2
inline-declaration	3.4	2.0
input-paramter-name	3.3	3.3
INSTANCE		2.5.1, 8.2.2
integer-conversion	7.0	5.1.1
integer-factor	5.1.1	5.1.1, 5.1.2, 5.1.3
integer-formula	5.1.1	4.4, 4.8, 5.1, 5.1.1, 5.2.1, 6.1, 6.3.2, 6.3.3, 6.3.4, 6.3.5
integer-function-call	5.1.1	5.1.1
integer-item-description	2.1.1.1	2.1.1.1
integer-literal	8.3.1	5.1.1, 8.3.1, 9.8
integer-machine-parameter	1.4	5.1.1
integer-primary	5.1.1	5.1.1, 5.1.2
integer-size	2.1.1.1	1.4, 2.1.1.1
integer-term	5.1.1	5.1.1, 5.1.3
integer-type-description	2.1.1.1	2.1.1, 7.0
integer-type-name	2.1.1.1	2.1.1.1, 7.0
integer-variable	5.1.1	5.1.1
INTERFERENCE		9.5

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
INTERRUPT		8.2.2
interference-control	9.5	9.5
interference-directive	9.5	9.0
INTPRECISION		1.4
intrinsic-function-call	6.3	6.3
ISBASE		9.8
isbase-directive	9.8	9.0
ITEM		2.1.1, 2.1.2.3, 2.1.2.4, 2.1.3, 8.2.2
item	6.1	6.1
item-declaration	2.1.1	2.1
item-dereference	6.1	6.1
item-name	2.1.1	2.1.1, 2.6, 4.2, 6.1
item-preset	2.1.6	2.1.1, 2.1.3
item-preset-value	2.1.6	2.1.6
item-type-declaration	2.2	2.2
item-type-description	2.1.1	2.1.1, 2.1.2.3, 2.1.1.4, 2.2, 3.2
item-type-name	2.2	2.1.1.1, 2.1.1.2, 2.1.1.3, 2.1.1.4, 2.1.1.5, 2.1.1.6, 2.1.1.7, 2.2
J		8.1, 8.3.2

<u>Construct</u>	<u>Definition</u>	<u>References</u>
K		8.1, 8.3.2
L		8.1, 8.3.2
LABEL		2.3, 8.2.2
label	4.0	1.2.3, 3.1, 4.0, 4.4
LAST		6.3.11, 8.2.2
LBOUND		6.3.9, 8.2.2
LEFTRIGHT		9.9
leftright-directive	9.9	9.0
letter	8.1	2.1.1.6, 2.4, 4.2, 8.1, 8.2, 8.2.1
letter-digit-\$-or-prime	8.2.1	8.2.1
letter-or-\$	8.2.1	8.2.1
LIKE		2.2, 8.2.2
like-option	2.2	2.2
LINKAGE		9.3
linkage-directive	9.3	9.0
LIST		9.7
LISTBOTH	9.7.2	9.0
LISTEXP	9.7.2	9.0
LISTINV	9.7.2	9.0
list-directive	9.7.1	9.0

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
literal	8.3	8.2
LOC		6.3.1, 8.2.2
loc-argument	6.3.1	6.3.1
location-specifier	2.1.2.4	2.1.2.4
loc-function	6.3.1	6.3
LOCSINWORD		1.4, 8.2.2
logical-continuation	5.2.1	5.2.1
logical-operand	5.2	5.2
logical-operator	8.2.3	8.2.3
loop-statement	4.2	4.0
loop-type	4.2	4.2
lower-bound	2.1.2.1	2.1.2.1, 4.4
lower-bound-option	2.1.2.1	2.1.2.1
M		2.1.2.3, 8.1, 8.3.2
machine-specific-function-call	6.3	6.3
machine-specific-procedure-call	4.5	4.5
main-program-module	1.2.3	1.1
mark	8.1	8.1
MAXBITS		1.4
MAXBYTES		1.4
MAXFIXED		1.4
MAXFIXEDPRECISION		1.4

<u>Construct</u>	<u>Definition</u>	<u>References</u>
MAXFLOAT		1.4
MAXFLOATPRECISION		1.4
MAXINT		1.4
MAXINTSIZE		1.4
MAXSIGDIGITS		1.4
MAXTABLESIZE		1.4
MINFIXED		1.4
MINFLOAT		1.4
MINFRACTION		1.4
MININT		1.4
MINRELPRECISION		1.4
MINSCALE		1.4
MINSIZE		1.4
MOD		8.2.2, 8.2.3
module	1.1	1.1
multiply-divide-or-mod	8.2.3	5.1.1, 8.2.3
multiply-or-divide	8.2.3	5.1.2, 5.1.3, 8.2.3
N		2.1.2.3, 8.1, 8.3.2
name	8.2.1	1.2.1, 1.2.3, 2.1.1, 2.1.1.6, 2.1.2.3, 2.1.3, 2.2, 2.4, 3.1, 3.2, 4.0, 8.2, 9.1, 9.4

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
named-bit-constant	5.2	5.2
named-character-constant	5.3	5.3
named-constant	6.2	5.1.1, 5.1.2, 5.1.3, 5.2, 5.3, 5.4, 5.5, 5.6
named-fixed-constant	5.1.3	5.1.3
named-floating-constant	5.1.2	5.1.2
named-integer-constant	5.1.1	5.1.1
named-floating-constant	5.1.2	5.1.2
named-pointer-constant	5.5	5.5
named-status-constant	5.4	5.4
named-table-constant	5.6	5.6
named-variable	6.1	6.1, 6.3.1
NENT		8.2.2
nested-block	4.5	4.5
NEW		8.2.2
NEXT		6.3.2, 8.2.2
next-argument	6.3.2	6.3.2
next-function	6.3.2	6.3
nbit	6.3.3	6.1, 6.3.3
nbyte	6.3.4	6.1, 6.3.4
NOLIST		9.7
nolist-directive	9.7.1	9.0
non-nested-subroutine	1.2.2	1.2.2, 1.2.3

<u>Construct</u>	<u>Definition</u>	<u>References</u>
NOT		5.2.1, 8.2.2, 8.2.3
NULL		8.2.2, 8.3.5
null-declaration	2.7	2.0, 2.1.2.3, 2.1.2.4, 2.1.4, 2.5.1, 2.5.2
null-statement	4.0	4.0
number	8.3.1	8.3.1
numeric-formula	5.1	4.2, 5.0, 6.3.6, 6.3.7
numeric-literal	8.3.1	8.3
NWDSSEN		6.3.10, 8.2.2
nwdsen-argument	6.3.10	6.3.10
nwdsen-function	6.3.10	6.3
O		8.1, 8.3.2
operator	8.2.3	8.2
OR		5.2.1, 8.2.2, 8.2.3
or-continuation	5.2	5.2
ORDER		9.11
order-directive	9.11	9.0
ordinary-entry-specifier	2.1.2.3	2.1.2
ordinary-table-body	2.1.2.3	2.1.2.3
ordinary-table-item-declaration	2.1.2.3	2.1.2.3
ordinary-table-options	2.1.2.3	2.1.2.3
other-character		8.1

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
output-paramter-name	3.3	3.3
OVERLAY		2.6, 8.2.2
overlay-address	2.6	2.6
overlay-declaration	2.6	2.0, 2.1.4
overlay-element	2.6	2.6
overlay-expression	2.6	2.6
overlay-string	2.6	2.6
P		2.1.1.7, 7.0, 8.1, 8.3.2
packing-specifier	2.1.2.3	2.1.2.3
PARALLEL		2.1.1.1, 8.2.2
paramter-binding	3.3	3.3
plus-or-minus	8.2.3	5.1.1, 5.1.2, 5.1.3, 8.2.3
pointer-conversion	7.0	5.5
pointer-formula	5.5	5.0, 5.2.2, 5.5, 6.1, 6.3.2
pointer-function-call	5.5	5.5
POS		2.1.2.4, 2.1.6, 2.6
pointer-item-description	2.1.1.7	2.1.1.7
pointer-item-name	6.1	6.1
pointer-literal	8.3.5	5.5
pointer-type-description	2.1.17	2.1.1, 7.0
pointer-type-name	2.1.1.7	2.1.1.7, 7.0

<u>Construct</u>	<u>Definition</u>	<u>References</u>
pointer-variable	5.5	5.5
precision	2.1.1.2	1.4, 2.1.1.2
preset-index-specifier	2.1.6	2.1.6
preset-values-option	2.1.6	2.1.6
PROC		3.1, 3.2, 8.2.2
procedure-body	3.1	3.1
procedure-call-statement	4.5	4.0
procedure-declaration	3.1	3.0
procedure-definition	3.1	3.0
procedure-heading	3.1	3.1
procedure-module	1.2.2	1.1
procedure-name	3.1	3.1, 3.3, 4.5, 6.3.1
PROTECTED		8.2.2
PROGRAM		1.2.3, 8.2.2
program-body	1.2.3	1.2.3
program-name	1.2.3	1.2.3
Q		8.1, 8.3.2
R		2.1.1.2, 8.1, 8.3.2
READONLY		8.2.2
real-literal	8.3.1	8.3.1
REARRANGE		9.9

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
rearrange-directive	9.9	9.0
REC		3.1, 8.2.2
REDUCIBLE		9.6
reducible-directive	9.6	9.0
REF		2.5.2, 8.2.2
ref-specification	2.5.2	2.5
ref-specification-choice	2.5.2	2.5.2
REGISTER		8.2.2
relational-expression	5.2.1	5.2
relational-operator	8.3.1	5.2.1, 8.2.3
RENT		3.1, 8.2.2
REP		7.0, 8.2.2
rep-conversion	7.0	6.1, 7.0
repetition-count	2.1.6	2.1.6
rep-function-variable	6.1	6.1
reserved-word	8.2.2	2.1.1.6, 8.2
RETURN		4.6, 8.2.2
return-statement	4.6	4.0
round-or-truncate	2.1.1.2	2.1.1.1, 2.1.1.2, 2.1.1.3
S		2.1.1.1, 7.0, 8.1, 8.3.2

<u>Construct</u>	<u>Definition</u>	<u>References</u>
scale-specifier	2.1.1.3	1.4, 2.1.1.3
separator	8.2.4	8.2
SGN		6.3.7, 8.2.2
shift-count	6.3.5	6.3.5
shift-direction	6.3.5	6.3.5
shift-function	6.3.5	6.3
SHIFTL		6.3.5, 8.2.2
SHIFTR		6.3.5, 8.2.2
sign	8.3.1	5.1.1, 5.1.2, 5.1.3. 8.3.1
SIGNAL		8.2.2
sign-function	6.3.7	6.3
simple-def	2.5.1	2.5.1
simple-ref	2.5.2	2.5.2
simple-statement	4.0	4.0
size-argument	6.3.8	6.3.8
size-function	6.3.8	6.3
size-type	6.3.8	6.3.8
SKIP		9.2.2
skip-directive	9.2.2	9.0
spacer	2.6	2.6
specified-entry-specifier	2.1.2.4	2.1.2
specified-item-description	2.1.2.4	2.1.2.4

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
specified-preset-sublist	2.1.6	2.1.6
specified-sublist	2.1.1.6	2.1.1.6
specified-table-body	2.1.2.4	2.1.2.4
specified-table -item-declaration	2.1.2.4	2.1.2.4
specified-table-options	2.1.2.4	2.1.2.4
START		1.2.1, 1.2.2, 1.2.3, 8.2.2
starting-bit	2.1.2.4	2.1.2.4
starting-word	2.1.2.4	2.1.2.4
statement	4.0	1.2.3, 3.1, 4.0, 4.2, 4.3, 4.4
statement-name	4.0	2.3, 3.3, 4.0, 4.5, 4.7, 6.3.1
statement-name-declaration	2.3	2.0, 2.5.1
STATIC		2.1.5, 8.2.2
STATUS		2.1.1.6, 8.2.2
status	2.1.1.6	2.1.1.6
status-constant	2.1.1.6	2.1.1.6, 5.4, 8.2
status-conversion	7.0	5.4
status-formula	5.4	4.4, 5.0, 5.4, 6.1, 6.3.11
status-function-call	5.4	5.4
status-inverse-argument	6.3.11	6.3.11
status-inverse-function	6.3.11	6.3

<u>Construct</u>	<u>Definition</u>	<u>References</u>
status-list	2.1.1.6	2.1.1.6
status-list-index	2.1.1.6	2.1.1.6
status-size	2.1.1.6	2.1.1.6
status-type-description	2.1.1.6	2.1.1, 7.0
status-type-name	2.1.1.6	2.1.1.6, 6.3.11, 7.0
status-variable	5.4	5.4
STOP		4.9, 8.2.2
stop-statement	4.9	4.0
structure-specifier	2.1.2.2	2.1.2, 2.2
subroutine-attribute	3.1	3.1, 3.2
subroutine-body	3.1	3.1, 3.2
subroutine-declaration	3.0	2.0, 2.5.2
subroutine-definition	3.0	1.2.2, 1.2.3, 3.1
subroutine-name	3.3	3.3, 3.4
subscript	6.1	6.1, 6.2
symbol	8.2	9.3
T		2.1.1.2, 2.1.2.2, 8.1, 8.3.2
TABLE		2.1.2, 2.1.3, 2.2, 8.2.2
table	6.1	6.1
table-conversion	7.0	5.6
table-declaration	2.1.2	2.1

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
table-dereference	6.1	6.1
table-description	2.1.2	2.1.2, 2.1.3
table-entry	6.1	6.1
table-formula	5.6	5.0, 5.6
table-item	6.1	6.1
table-item-name	2.1.2.3	2.1.2.3, 2.1.2.4, 6.1, 6.2
table-name	2.1.2	2.1.2, 2.6, 6.1, 6.3.9, 6.3.10
table-preset	2.1.6	2.1.2, 2.1.2.3, 2.1.2.4
table-preset-list	2.1.6	2.1.6
table-type-declaration	2.2	2.2
table-type-name	2.2	2.1.1.7, 2.1.2, 2.2, 6.3.10, 7.0
table-type-specifier	2.2	2.2
table-variable	5.6	5.6
TERM		1.2.1, 1.2.3, 8.2.2
THEN		4.2, 8.2.2
then-phrase	4.2	4.2
TO		8.2.2
TRACE		9.4
trace-control	9.4	9.4
trace-directive	9.4	9.0
TRUE		8.2.2, 8.3.3

<u>Construct</u>	<u>Definition</u>	<u>References</u>
TYPE		2.2, 8.2.2
type-declaration	2.2	2.0
type-name	2.1.1.7	2.1.1.7
U		2.1.1.1, 7.0, 8.1, 8.3.2
UBOUND		6.3.9, 8.2.2
UPDATE		8.2.2
upper-bound	2.1.2.1	2.1.2.1, 4.4
user-defined-function-call	6.3	6.3
user-defined-procedure-call	4.5	4.5
V		2.1.1.6, 2.1.2.4, 8.1, 8.3.2
variable	6.1	4.1, 4.5, 5.1.1, 5.1.2, 5.1.3, 5.2, 5.3, 5.4, 5.5, 5.6, 6.1
variable-list	4.1	4.1
W		2.1.2.4, 2.6, 8.1
which-bound	6.3.9	6.3.9
WHILE		4.2, 8.2.2
while-clause	4.2	4.2
while-phrase	4.2	4.2

MIL-STD-1589B (USAF)
06 June 1980

<u>Construct</u>	<u>Definition</u>	<u>References</u>
WITH		8.2.2
WORDSIZE		6.3.8, 8.2.2
words-per-entry	2.1.2.4	2.1.2.4
WRITEONLY		8.2.2
X		8.1
XOR		5.2.1, 8.2.2, 8.2.3
xor-continuation	5.2	5.2
Y		8.1
Z		2.1.1.2, 8.1
ZONE		8.2.2

MIL-STD-1750A (USAF)
2 July 1980
SUPERSEDING
MIL-STD-1750 (USAF)
21 February 1979

MILITARY STANDARD
SIXTEEN-BIT COMPUTER INSTRUCTION SET ARCHITECTURE



FSC IPSC

MIT STD 1750A (USAF)
2 July 1980

DEPARTMENT OF DEFENSE
Washington DC 20360

Sixteen-bit Computer Instruction Set Architecture MIT STD 1750A (USAF)

1. This Military Standard is approved for use by the Department of the Air Force, and is available for use by all Departments and Agencies of the Department of Defense.

2. Beneficial comments (recommendations, additions, deletions) and any pertinent data which may be of use in improving this document should be addressed to: Aeronautical Systems Division, Attn: ASD/ENESS, Wright Patterson Air Force Base, Ohio 45433, by using the self-addressed Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document or by letter.

CONTENTS

Paragraph		Page
1	SCOPE AND PURPOSE - - - - -	1
1.1	Scope - - - - -	1
1.2	Purpose - - - - -	1
1.3	Applicability - - - - -	1
1.4	Benefits - - - - -	1
2	REFERENCED DOCUMENTS - - - - -	1
3	DEFINITIONS - - - - -	1
3.1	Accumulator - - - - -	1
3.2	Address - - - - -	1
3.3	Arithmetic logic unit (ALU) - - - - -	1
3.4	Avionics - - - - -	1
3.5	Base register - - - - -	1
3.6	Bit - - - - -	1
3.7	Byte - - - - -	1
3.8	Central processing unit (CPU) - - - - -	1
3.9	Control unit - - - - -	2
3.10	General purpose register - - - - -	2
3.11	Index register - - - - -	2
3.12	Input/output (I/O) - - - - -	2
3.13	Instruction - - - - -	2
3.14	Instruction counter (IC) - - - - -	2
3.15	Instruction set architecture (ISA) - - - - -	2
3.16	Interrupt - - - - -	2
3.17	Memory - - - - -	2
3.18	Operation code (OPCODE) - - - - -	2
3.19	Operand - - - - -	2
3.20	Page register - - - - -	2
3.21	Programmed input/output (PIO) - - - - -	2
3.22	Register - - - - -	2
3.23	Register transfer language (RTL) - - - - -	2
3.24	Reserved - - - - -	2
3.25	Spare - - - - -	2
3.26	Stack - - - - -	2
3.27	Stack pointer - - - - -	3
3.28	Status word register - - - - -	3
3.29	Word - - - - -	3
4	GENERAL REQUIREMENTS - - - - -	3
4.1	Data formats - - - - -	3
4.1.1	Single precision fixed point data - - - - -	3
4.1.2	Double precision fixed point data - - - - -	4
4.1.3	Fixed point operands - - - - -	4
4.1.4	Results on fixed point overflow - - - - -	4
4.1.5	Floating point data - - - - -	4
4.1.6	Extended precision floating point data - - - - -	5
4.1.7	Floating point operands - - - - -	6

HD-510-1750A (USA1)

2 July 1980

4.1.8	Truncation of floating point results	-	-	-	-	-	-	-	-	6
4.1.9	Results of division	-	-	-	-	-	-	-	-	6
4.2	Instruction formats	-	-	-	-	-	-	-	-	7
4.2.1	Register-to-register format	-	-	-	-	-	-	-	-	7
4.2.2	Instruction counter relative format	-	-	-	-	-	-	-	-	7
4.2.3	Base relative format	-	-	-	-	-	-	-	-	7
4.2.4	Base relative indexed format	-	-	-	-	-	-	-	-	7
4.2.5	Long instruction format	-	-	-	-	-	-	-	-	8
4.2.6	Immediate opcode extension format	-	-	-	-	-	-	-	-	8
4.3	Addressing modes	-	-	-	-	-	-	-	-	8
4.3.1	Register direct (R)	-	-	-	-	-	-	-	-	8
4.3.2	Memory direct (D)	-	-	-	-	-	-	-	-	8
4.3.3	Memory direct-indexed (DX)	-	-	-	-	-	-	-	-	8
4.3.4	Memory indirect (I)	-	-	-	-	-	-	-	-	10
4.3.5	Memory indirect with pre-indexing (IX)	-	-	-	-	-	-	-	-	10
4.3.6	Immediate long (IM)	-	-	-	-	-	-	-	-	10
4.3.7	Immediate short (IS)	-	-	-	-	-	-	-	-	10
4.3.7.1	Immediate short positive (ISP)	-	-	-	-	-	-	-	-	10
4.3.7.2	Immediate short negative (ISN)	-	-	-	-	-	-	-	-	10
4.3.8	Instruction counter relative (ICR)	-	-	-	-	-	-	-	-	10
4.3.9	Base relative (B)	-	-	-	-	-	-	-	-	10
4.3.10	Base relative-indexed (BX)	-	-	-	-	-	-	-	-	10
4.3.11	Special (S)	-	-	-	-	-	-	-	-	10
4.4	Registers and support features	-	-	-	-	-	-	-	-	10
4.4.1	General registers	-	-	-	-	-	-	-	-	10
4.4.2	Special registers	-	-	-	-	-	-	-	-	11
4.4.2.1	Instruction counter (IC)	-	-	-	-	-	-	-	-	11
4.4.2.2	Status word (SW)	-	-	-	-	-	-	-	-	11
4.4.2.3	Fault register (FT)	-	-	-	-	-	-	-	-	12
4.4.2.4	Interrupt mask (MK)	-	-	-	-	-	-	-	-	13
4.4.2.5	Pending interrupt register (PI)	-	-	-	-	-	-	-	-	13
4.4.2.6	Input/output interrupt code registers (IOIC)(optional)	-	-	-	-	-	-	-	-	13
4.4.2.7	Page registers (optional)	-	-	-	-	-	-	-	-	13
4.4.2.8	Memory fault status register (MFSR) (optional)	-	-	-	-	-	-	-	-	13
4.4.3	Stack	-	-	-	-	-	-	-	-	14
4.4.4	Processor initialization	-	-	-	-	-	-	-	-	14
4.4.4.1	Processor reset state	-	-	-	-	-	-	-	-	14
4.4.4.2	Power up	-	-	-	-	-	-	-	-	14
4.4.5	Interval timers (optional)	-	-	-	-	-	-	-	-	14
4.5	Memory	-	-	-	-	-	-	-	-	15
4.5.1	Memory addressing	-	-	-	-	-	-	-	-	15
4.5.1.1	Memory addressing arithmetic	-	-	-	-	-	-	-	-	15
4.5.1.2	Memory addressing boundary constraints	-	-	-	-	-	-	-	-	15
4.5.2	Expanded memory addressing (optional)	-	-	-	-	-	-	-	-	15
4.5.2.1	Group selection	-	-	-	-	-	-	-	-	15
4.5.2.2	Page register word format	-	-	-	-	-	-	-	-	15
4.5.2.3	Partial implementations of expanded memory addressing	-	-	-	-	-	-	-	-	18
4.5.3	Memory parity (optional)	-	-	-	-	-	-	-	-	18
4.5.4	Memory block protect (optional)	-	-	-	-	-	-	-	-	18
4.5.5	References to unimplemented memory	-	-	-	-	-	-	-	-	18
4.5.6	Start up ROM (optional)	-	-	-	-	-	-	-	-	18
4.5.7	Reserved memory locations	-	-	-	-	-	-	-	-	18
4.6	Interrupt control	-	-	-	-	-	-	-	-	18

4.6.1	Interrupts - - - - -	-	-	-	-	-	-	-	-	-	18
4.6.1.1	Interrupt acceptance - - - - -	-	-	-	-	-	-	-	-	-	18
4.6.1.2	Interrupt software control - - - - -	-	-	-	-	-	-	-	-	-	18
4.6.1.3	Interrupt priority definitions - - - - -	-	-	-	-	-	-	-	-	-	21
4.6.1.4	Interrupt vectoring mechanism - - - - -	-	-	-	-	-	-	-	-	-	21
4.7	Input/output - - - - -	-	-	-	-	-	-	-	-	-	21
4.7.1	Input - - - - -	-	-	-	-	-	-	-	-	-	22
4.7.2	Output - - - - -	-	-	-	-	-	-	-	-	-	22
4.7.3	Input/output commands - - - - -	-	-	-	-	-	-	-	-	-	22
4.7.4	Input/output command partitioning - - - - -	-	-	-	-	-	-	-	-	-	22
4.7.5	Input/output interrupts (optional) - - - - -	-	-	-	-	-	-	-	-	-	22
4.7.6	Dedicated I/O memory locations - - - - -	-	-	-	-	-	-	-	-	-	22
4.8	Instructions - - - - -	-	-	-	-	-	-	-	-	-	22
4.8.1	Invalid instructions - - - - -	-	-	-	-	-	-	-	-	-	22
4.8.2	Mnemonic conventions - - - - -	-	-	-	-	-	-	-	-	-	22
4.8.3	Instruction matrix - - - - -	-	-	-	-	-	-	-	-	-	23
4.8.4	Instruction set notation - - - - -	-	-	-	-	-	-	-	-	-	23
5	DETAILED REQUIREMENTS - - - - -	-	-	-	-	-	-	-	-	-	29
5.1	Execute input/output - - - - -	-	-	-	-	-	-	-	-	-	29
5.2	Vectored input/output - - - - -	-	-	-	-	-	-	-	-	-	33
5.3	Set bit - - - - -	-	-	-	-	-	-	-	-	-	34
5.4	Reset bit - - - - -	-	-	-	-	-	-	-	-	-	35
5.5	Test bit - - - - -	-	-	-	-	-	-	-	-	-	36
5.6	Test and set bit - - - - -	-	-	-	-	-	-	-	-	-	37
5.7	Set variable bit in register - - - - -	-	-	-	-	-	-	-	-	-	38
5.8	Reset variable bit in register - - - - -	-	-	-	-	-	-	-	-	-	39
5.9	Test variable bit in register - - - - -	-	-	-	-	-	-	-	-	-	40
5.10	Shift left logical - - - - -	-	-	-	-	-	-	-	-	-	41
5.11	Shift right logical - - - - -	-	-	-	-	-	-	-	-	-	42
5.12	Shift right arithmetic - - - - -	-	-	-	-	-	-	-	-	-	43
5.13	Shift left cyclic - - - - -	-	-	-	-	-	-	-	-	-	44
5.14	Double shift left logical - - - - -	-	-	-	-	-	-	-	-	-	45
5.15	Double shift right logical - - - - -	-	-	-	-	-	-	-	-	-	46
5.16	Double shift right arithmetic - - - - -	-	-	-	-	-	-	-	-	-	47
5.17	Double shift left cyclic - - - - -	-	-	-	-	-	-	-	-	-	48
5.18	Shift logical, count in register - - - - -	-	-	-	-	-	-	-	-	-	49
5.19	Shift arithmetic, count in register - - - - -	-	-	-	-	-	-	-	-	-	50
5.20	Shift cyclic, count in register - - - - -	-	-	-	-	-	-	-	-	-	51
5.21	Double shift logical, count in register - - - - -	-	-	-	-	-	-	-	-	-	52
5.22	Double shift arithmetic, count in register - - - - -	-	-	-	-	-	-	-	-	-	53
5.23	Double shift cyclic, count in register - - - - -	-	-	-	-	-	-	-	-	-	54
5.24	Jump on condition - - - - -	-	-	-	-	-	-	-	-	-	55
5.25	Jump to subroutine - - - - -	-	-	-	-	-	-	-	-	-	57
5.26	Subtract one and jump - - - - -	-	-	-	-	-	-	-	-	-	58
5.27	Branch unconditionally - - - - -	-	-	-	-	-	-	-	-	-	59
5.28	Branch if equal to (zero) - - - - -	-	-	-	-	-	-	-	-	-	60
5.29	Branch if less than (zero) - - - - -	-	-	-	-	-	-	-	-	-	61
5.30	Branch to executive - - - - -	-	-	-	-	-	-	-	-	-	62
5.31	Branch if less than or equal to (zero) - - - - -	-	-	-	-	-	-	-	-	-	63
5.32	Branch if greater than (zero) - - - - -	-	-	-	-	-	-	-	-	-	64
5.33	Branch if not equal to (zero) - - - - -	-	-	-	-	-	-	-	-	-	65

HLI STD 1750A (USA1)
2 July 1980

5.35	Load status - - - - -	67
5.36	Stack IC and jump to subroutine - - - - -	68
5.37	Unstack IC and return from subroutine - - - - -	69
5.38	Single precision load - - - - -	70
5.39	Double precision load - - - - -	72
5.40	Load multiple registers - - - - -	73
5.41	Extended precision floating point load - - - - -	74
5.42	Load from upper byte - - - - -	75
5.43	Load from lower byte - - - - -	76
5.44	Pop multiple registers off the stack - - - - -	77
5.45	Single precision store - - - - -	78
5.46	Store a non-negative constant - - - - -	79
5.47	Move multiple words, memory-to-memory - - - - -	80
5.48	Double precision store - - - - -	81
5.49	Store register through mask - - - - -	82
5.50	Store multiple registers - - - - -	83
5.51	Extended precision floating point store - - - - -	84
5.52	Store into upper byte - - - - -	85
5.53	Store into lower byte - - - - -	86
5.54	Push multiple registers onto the stack - - - - -	87
5.55	Single precision integer add - - - - -	89
5.56	Increment memory by a positive integer - - - - -	91
5.57	Single precision absolute value of register - - - - -	92
5.58	Double precision absolute value of register - - - - -	93
5.59	Double precision integer add - - - - -	94
5.60	Floating point add - - - - -	95
5.61	Extended precision floating point add - - - - -	97
5.62	Floating point absolute value of register - - - - -	98
5.63	Single precision integer subtract - - - - -	99
5.64	Decrement memory by a positive integer - - - - -	101
5.65	Single precision negate register - - - - -	102
5.66	Double precision negate register - - - - -	103
5.67	Double precision integer subtract - - - - -	104
5.68	Floating point subtract - - - - -	105
5.69	Extended precision floating point subtract - - - - -	107
5.70	Floating point negate register - - - - -	108
5.71	Single precision integer multiply with 16-bit product - - - - -	109
5.72	Single precision integer multiply with 32-bit product - - - - -	110
5.73	Double precision integer multiply - - - - -	111
5.74	Floating point multiply - - - - -	112
5.75	Extended precision floating point multiply - - - - -	114
5.76	Single precision integer divide with 16-bit dividend - - - - -	116
5.77	Single precision integer divide with 32-bit dividend - - - - -	117
5.78	Double precision integer divide - - - - -	118
5.79	Floating point divide - - - - -	119
5.80	Extended precision floating point divide - - - - -	121
5.81	Inclusive logical OR - - - - -	122
5.82	Logical AND - - - - -	123
5.83	Exclusive logical OR - - - - -	124
5.84	Logical NAND - - - - -	125
5.85	Convert floating point to 16 bit integer - - - - -	126
5.86	Convert 16 bit integer to floating point - - - - -	127
5.87	Convert extended precision floating point to 32-bit integer - - - - -	128

5.88	Convert 32-bit integer to extended precision floating point	-	129
5.89	Exchange bytes in register	-	130
5.90	Exchange words in registers	-	131
5.91	Single precision compare	-	132
5.92	Compare between limits	-	133
5.93	Double precision compare	-	134
5.94	Floating point compare	-	135
5.95	Extended precision floating point compare	-	136
5.96	No operation	-	137
5.97	Break point	-	138
	INDEX	-	140

MIL-STD-1750A (USAF)
2 July 1980

FIGURES

<u>Figure</u>		<u>Page</u>
1	Expanded memory mapping diagram - - - - -	16
2	Interrupt system flowchart - - - - -	20
3	Interrupt vectoring system - - - - -	21

TABLES

<u>Table</u>		<u>Page</u>
I	Single precision fixed point numbers - - - - -	3
II	Double precision fixed point numbers - - - - -	4
III	32-bit floating point numbers - - - - -	5
IV	48-bit extended floating point numbers - - - - -	6
V	Addressing modes and instruction word format - - - - -	9
VI	Processor reset state - - - - -	14
VII	AL code to access key mapping - - - - -	17
VIII	Interrupt definitions - - - - -	19
IX	Input/output channel groups - - - - -	23
X	Operation code matrix - - - - -	27
XI	Extended operation codes - - - - -	28

1 SCOPE AND PURPOSE

1.1 Scope. This standard defines the instruction set architecture (ISA) for airborne computers. It does not define specific implementation details of a computer.

1.2 Purpose. The purpose of this document is to establish a uniform instruction set architecture for airborne computers which shall be used in Air Force avionic weapon systems.

1.3 Applicability. This standard is intended to be used to define only the ISA of airborne computers. System-unique requirements such as speed, weight, power, additional input/output commands, and environmental operating characteristics are defined in the computer specification for each computer. Application is not restricted to any particular avionic function or specific hardware implementation by this standard. Generally, the ISA is applicable to, and shall be used for, computers that perform such functions as moderate accuracy navigation, computed air release points, weapon delivery, air rendezvous, stores management, aircraft guidance, and aircraft management. This standard is not restricted to implementations of "stand-alone" computers such as a mission computer or a fire control computer. Application to the entire range of avionics functions is encouraged such as stability and control, display processing and control, thrust management, and electrical power control.

1.4 Benefits. The expected benefits of this standard ISA are the use and re-use of available support software such as compilers and instruction level simulators. Other benefits may also be achieved such as: (a) reduction in total support software gained by the use of the standard ISA for two or more computers in a weapon system, and (b) software development independent of hardware development.

2 REFERENCED DOCUMENTS

Not applicable.

3 DEFINITIONS

3.1 Accumulator. A register in the arithmetic logic unit used for intermediate storage, algebraic sums and other arithmetic and logical results.

3.2 Address. A number which identifies a location in memory where information is stored.

3.3 Arithmetic logic unit (ALU). That portion of hardware in the central processing unit in which arithmetic and logical operations are performed.

3.4 Avionics. All the electronic and electromechanical systems and subsystems (hardware and software) installed in an aircraft or attached to it. Avionics systems interact with the crew or other aircraft systems in these functional areas: communications, navigation, weapons delivery, identification, instrumentation, electronic warfare, reconnaissance, flight control, engine control, power distribution, and support equipment.

3.5 Base register. Any general register used to provide the base address portion of the derived address for instructions using the base relative or base relative-indexed addressing modes.

3.6 Bit. Contraction of binary digit; may be either zero or one. In information theory, a binary digit is equal to one binary decision or the designation of one of two possible values or states of anything used to store or convey information.

3.7 Byte. A group of eight binary digits.

3.8 Central processing unit (CPU). That portion of a computer that controls and performs the execution of instructions.

- 3.9 Control unit. That portion of hardware in the CPU that directs sequence of operations, interprets coded instructions, and initiates proper commands to other parts of the computer.
- 3.10 General purpose register. A register that may be used for arithmetic and logical operations, indexing, shifting, input/output, and general storage of temporary data.
- 3.11 Index register. A register that contains a quantity for modification of an address without permanently modifying the address.
- 3.12 Input/output (I/O). That portion of a computer which interfaces to the external world.
- 3.13 Instruction. A program code which tells the computer what to do.
- 3.14 Instruction counter (IC). A register in the CPU that holds the address of the next instruction to be executed.
- 3.15 Instruction set architecture (ISA). The attributes of a digital computer as seen by a machine (assembly) language programmer. ISA includes the processor and input/output instruction sets, their formats, operation codes, and addressing modes; memory management and partitioning if accessible to the machine language programmer, the speed of accessible clocks; interrupt structure; and the manner of use and format of all registers and memory locations that may be directly manipulated or tested by a machine language program. This definition excludes the time or speed of any operation, internal computer partitioning, electrical and physical organization, circuits and components of the computer, manufacturing technology, memory organization, memory cycle time, and memory bus widths.
- 3.16 Interrupt. A special control signal that suspends the normal flow of the processor operations and allows the processor to respond to a logically unrelated or unpredictable event.
- 3.17 Memory. That portion of a computer that holds data and instructions and from which they can be accessed at a later time.
- 3.18 Operation code (OPCODE). That part of an instruction that defines the machine operation to be performed.
- 3.19 Operand. That part of an instruction that specifies the address of the source, the address of the destination, or the data itself on which the processor is to operate.
- 3.20 Page register. A register which is used to supply additional address bits in paged memory addressing schemes.
- 3.21 Programmed input/output (PIO). A type of I/O channel that allows program control of information transfer between the computer and an external device.
- 3.22 Register. A device in the CPU for the temporary storage of one or more words to facilitate arithmetical, logical, or transfer operations.
- 3.23 Register transfer language (RTL). A language used to describe operations (upon registers) which are caused by the execution of each instruction.
- 3.24 Reserved. Must not be used.
- 3.25 Spare. A framework for usage is defined by the standard with particulars to be defined by the application requirements.
- 3.26 Stack. A sequence of memory locations in which data may be stored and retrieved on a last-in-first-out (LIFO) basis.

3.27 Stack pointer. A register that points to the last item on the stack.

3.28 Status word register. A register whose state is defined by some prior event occurrence in the computer.

3.29 Word. Sixteen bits.

4 GENERAL REQUIREMENTS

4.1 Data formats. The instruction set shall support 16-bit fixed point single precision, 32-bit fixed point double precision, 32-bit floating point and 48-bit floating point extended precision data in 2's complement representation.

4.1.1 Single precision fixed point data. Single precision 16-bit fixed point data shall be represented as a 16-bit 2's complement integer number with the most significant bit (MSB) as the sign bit:

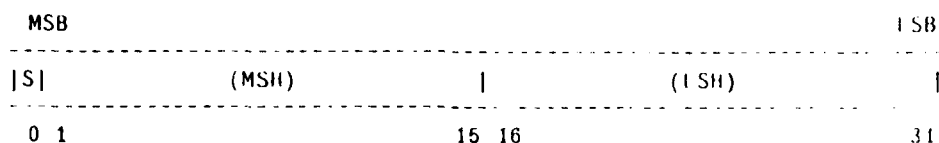
MSB	LSB
S	
0 1	15

Examples of single precision fixed point numbers are shown in table I.

TABLE I. Single precision fixed point numbers

Integer	16-Bit Hexadecimal Word
32767	7 F F F
16384	4 0 0 0
4096	1 0 0 0
2	0 0 0 2
1	0 0 0 1
0	0 0 0 0
-1	F F F F
-2	F F F E
-4096	F 0 0 0
-16384	C 0 0 0
-32767	8 0 0 1
-32768	8 0 0 0

4.1.2 Double precision fixed point data. Double precision 32-bit fixed point data shall be represented as a 32-bit 2's complement integer number with the most significant bit (MSB) of the first word as the sign bit.



Examples of machine representation for double precision fixed point numbers are shown in table II.

TABLE II. Double precision fixed point numbers

Integer	32-Bit Hexadecimal Word
2,147,483,647	7 F F F F F F F
1,073,741,824	4 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 2
1	0 0 0 0 0 0 0 1
0	0 0 0 0 0 0 0 0
-1	F F F F F F F F
-2	F F F F F F F E
-1,073,741,825	C 0 0 0 0 0 0 0 0
-2,147,483,647	8 0 0 0 0 0 0 1
-2,147,483,648	8 0 0 0 0 0 0 0

4.1.3 Fixed point operands. All operands for fixed point adds, subtracts, multiplies and divides are integer. A fixed point overflow shall be defined as arithmetic overflow if the result is greater than $7FFF_{16}$ or less than 8000_{16} for single precision and greater than $7FFF\ FFFF_{16}$ or less than $8000\ 0000_{16}$ for double precision.

4.1.4 Results on fixed point overflow. On fixed point operations which cause overflow, the operation shall be performed to completion as if the MSBs are present and the 16 LSBs for single precision or the 32 LSBs for double precision shall be retained in the proper register(s). Division by zero shall produce a fixed point overflow and return results of all zeros.

4.1.5 Floating point data. Floating point data shall be represented as a 32-bit quantity consisting of a 24-bit 2's complement mantissa and an 8-bit 2's complement exponent.

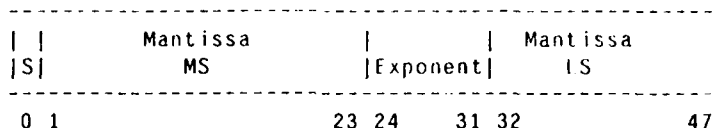
MSB	LSB	MSB	LSB
S	Mantissa	Exponent	
0 1		23 24	31

Floating point numbers are represented as a fractional mantissa times 2 raised to the power of the exponent. All floating point numbers are assumed normalized or floating point zero at the beginning of a floating point operation and the results of all floating point operations are normalized (a normalized floating point number has the sign of the mantissa and the next bit of opposite value) or floating point zero. A floating point zero is defined as 0000 0000₁₆, that is, a zero mantissa and a zero exponent (00₁₆). An extended floating point zero is defined as 0000 0000 0000₁₆, that is, a zero mantissa and a zero exponent. Some examples of the machine representation for 32-bit floating point numbers are shown in table III.

TABLE III. 32-bit floating point numbers

Decimal Number	Hexadecimal Notation	
	Mantissa	EXP
0.9999998×2^{127}	7FFF FF	7F
0.5×2^{127}	4000 00	7F
0.625×2^4	5000 00	04
0.5×2^1	4000 00	01
0.5×2^0	4000 00	00
0.5×2^{-1}	4000 00	FF
0.5×2^{-128}	4000 00	80
0.0×2^0	0000 00	00
-1.0×2^0	8000 00	00
$-0.5000001 \times 2^{-128}$	BFFF FF	80
-0.7500001×2^4	9FFF FF	04

4.1.6 Extended precision floating point data. Extended floating point data shall be represented as a 48-bit quantity consisting of a 40-bit 2's complement mantissa and an 8-bit 2's complement exponent. The exponent bits 24 to 31 lay between the split mantissa bits 0 to 23 and bits 32 to 47. The most significant bit of the mantissa is the sign bit 0, and the least significant bit of the mantissa is bit 47.



Some examples of the machine representation of 48-bit extended floating point numbers are shown in table IV.

TABLE IV. 48-bit extended floating point numbers

Decimal Number	Hexadecimal Notation		
	Mantissa (MS)	Exp	Mantissa (LS)
0.5×2^{127}	400000	7F	0000
0.5×2^0	400000	00	0000
0.5×2^{-1}	400000	FF	0000
0.5×2^{-128}	400000	80	0000
-1.0×2^{127}	800000	7F	0000
-1.0×2^0	800000	00	0000
-1.0×2^{-1}	800000	FF	0000
-1.0×2^{-128}	800000	80	0000
0.0×2^0	000000	00	0000
-0.75×2^{-1}	A00000	FF	0000

For both floating point and extended floating point numbers, an overflow is defined as an exponent overflow and an underflow is defined as an exponent underflow.

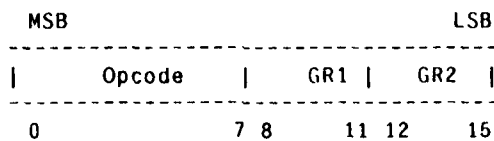
4.1.7 Floating point operands. All operands for floating point instructions must be normalized or a floating point zero. A floating point overflow shall be defined as exponent overflow if the exponent is greater than $7F_{16}$. The results of an operation which causes a floating point overflow shall be the largest positive number if the sign of the resulting mantissa was positive, or shall be the smallest negative number if the sign of the resulting mantissa was negative. Underflow shall be defined as exponent underflow if the exponent is less than 80_{16} . The results of an operation which causes a floating point underflow shall be floating point zero. Separate interrupts are set for overflow and underflow. Only the floating point instructions shall set the underflow interrupt.

4.1.8 Truncation of floating point results. All floating point results shall be truncated toward negative infinity.

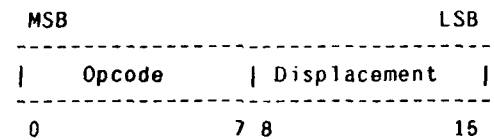
4.1.9 Results of division. The sign of any non-zero remainder is the same as the dividend for all division instructions; the remainder is only accessible for single precision integer divides with 16 bit dividends and for single precision integer divides with 32 bit dividends.

4.2 Instruction formats. Six basic instruction formats shall support 16 and 32-bit instructions. The operation code (opcode) shall normally consist of the 8 most significant bits of the instruction.

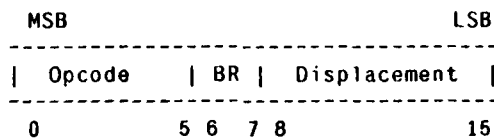
4.2.1 Register-to-register format. The register-to-register format is a 16-bit instruction consisting of an 8-bit opcode and two 4-bit general register (GR) fields that typically specify any of 16 general registers. In addition, these fields may contain a shift count, condition code, opcode extension, bit number, or the operand for immediate short instructions.



4.2.2 Instruction counter relative format. The Instruction Counter (IC) Relative Format is a 16-bit instruction consisting of an 8-bit opcode and an 8-bit displacement field.



4.2.3 Base relative format. The base relative instruction format is a 16-bit instruction consisting of a 6-bit opcode, a 2-bit base register field and an 8-bit displacement field. The base register (BR) field allows the designation of one of four different registers.



BR = 0 implies general register 12

BR = 1 implies general register 13

BR = 2 implies general register 14

BR = 3 implies general register 15

4.2.4 Base relative indexed format. The base relative indexed instruction format is a 16-bit instruction consisting of a 6-bit opcode, a 2-bit base register field, a 4-bit opcode extension and a 4-bit index register field. The base register (BR) field allows the designation of one of four different base registers and the index register (RX) field allows the designation of one of fifteen different index registers.

MIL STD-1750A (USAF)
2 July 1980

MSB								LSB							
Opcode				BR Op.Ex.				RX							
0				5 6 7 8				11 12				15			

BR = 0 implies general register 12

BR = 1 implies general register 13

BR = 2 implies general register 14

BR = 3 implies general register 15

RX = 0 implies no indexing

4.2.5 Long instruction format. The Long Instruction Format is a 32-bit instruction consisting of an 8-bit opcode, a 4-bit general register field, a 4-bit index register field and a 16-bit address field.

MSB																LSB								
Opcode								GR1				RX				16-Bit Address Field								
0								7 8				11 12				15 16								31

Typically, GR1 is one of the 16 general registers on which the instruction is performing the operation. RX is one of the 15 general registers being used as an index register. The 16-bit address field is either a full 16-bit memory address or a 16-bit operand if the instruction specifies immediate addressing.

4.2.6 Immediate opcode extension format. The immediate opcode extension format is a 32-bit instruction consisting of an 8-bit opcode, a 4-bit general register field, a 4-bit opcode extension and a 16-bit data field. Typically, GR1 is one of the 16 general registers on which the instruction is performing the operation. Op. Ex. is an opcode extension.

MSB															LSB
Opcode				GR1				Op.Ex.				16-Bit Immediate Data			
0				7 8				11 12				15 16			31

4.3 Addressing modes. Table V specifies the instruction word format, the Derived Address (DA), and the Derived Operand (DO) for each addressing mode that shall be implemented. The smallest addressable memory word is 16 bits; hence, the 16-bit address fields allow direct addressing of 64K (65,536) words. There is no restriction on the location of double word operands in memory.

4.3.1 Register direct (R). An addressing mode in which the instruction specified register contains the required operand. (With the exception of this address mode, DA denotes a memory address.)

4.3.2 Memory direct (D). An addressing mode in which the instruction contains the memory address of the operand.

4.3.3 Memory direct-indexed (IX). An addressing mode in which the memory address of the required operand is specified by the sum of the content of an index register and the instruction address field. Registers R1, R2, ..., R15 may be specified for indexing.

TABLE V. Addressing modes and instruction word format

MODE	FORMAT	DERIVED OPERAND (D.O.)				DERIVED ADDRESS (D.A.)				NOTES
		S.P.	PL and D.P.	S.P.	PL and D.P.	S.P.	PL and D.P.	S.P.	PL and D.P.	
1. Register Direct	0 1 0 1 1 1 1 1 5 O.C. RA RE	(RS)	(RR, RR+1)	RB	RR, RR+1	RB	RR, RR+1	RB	RR, RR+1	Fast add, subtract, multiply, divide, and other operations on three operands located at RA, RA+1, and RA+2.
2. Memory Direct	0 7 8 11 12 15 16 31 O.C. RA RA RA	[A] [A+(RR)]	[A+(RR), A+1+(RR)]	A A+(RR)	A, A+1 A+(RR), A+1+(RR)	A A+(RR)	A, A+1 A+(RR), A+1+(RR)	A A+(RR)	A, A+1 A+(RR), A+1+(RR)	
3. Memory Indirect	0 7 8 11 12 15 16 31 O.C. RA RA RA	[A] [A+(RR)]	[A+(RR), A+1+(RR)]	[A] [A+(RR)]	[A], [A+1] [A+(RR)], [A+1+(RR)]	[A] [A+(RR)]	[A], [A+1] [A+(RR)], [A+1+(RR)]	[A] [A+(RR)]	[A], [A+1] [A+(RR)], [A+1+(RR)]	
4. Immediate Long	0 7 8 11 12 15 16 31 O.C. RA RA RA	---	---	---	---	---	---	---	---	
5. Immediate Short	0 7 8 11 12 15 16 31 O.C. RA RA RA	---	---	---	---	---	---	---	---	Base registers: BR0, BR1, BR2, BR3, BR4, and BR5. G4DU, G5S.
6. IC Relative	0 7 8 11 12 15 16 31 O.C. RA RA RA	---	---	---	---	---	---	---	---	
7. Base Relative	0 5 6 7 8 11 12 15 16 31 O.C. RA RA RA	[DU+(RR)]	[DU+(RR), DU+1+(RR)]	DU+(RR)	DU+(RR), DU+1+(RR)	DU+(RR)	DU+(RR), DU+1+(RR)	DU+(RR)	DU+(RR), DU+1+(RR)	
8. Not Indirect	0 5 6 7 8 11 12 15 16 31 O.C. RA RA RA	[DU+(RR)]	[DU+(RR), DU+1+(RR)]	DU+(RR)	DU+(RR), DU+1+(RR)	DU+(RR)	DU+(RR), DU+1+(RR)	DU+(RR)	DU+(RR), DU+1+(RR)	

2 July 1980

4.3.4 Memory indirect (I). An addressing mode in which the instruction specified memory address contains the address of the required operand.

4.3.5 Memory indirect with pre-indexing (IX). An addressing mode in which the sum of the content of a specified index register and the instruction address field is the address of the address of the required operand. Registers R1, R2, ..., R15 may be specified for pre-indexing.

4.3.6 Immediate long (IM). There shall be two methods of Immediate Long addressing: one which allows indexing and one which does not. The indexable form of immediate addressing is defined in table V. If the specified index register, RX, is not equal to zero, the content of RX is added to the immediate field to form the required operand; otherwise the immediate field contains the required operand.

4.3.7 Immediate short (IS). An addressing mode in which the required (4-bit) operand is contained within the (16-bit) instruction. There shall be two methods of Immediate Short addressing: one which interprets the content of the immediate field as positive data, and a second which interprets the content of immediate field as negative data.

4.3.7.1 Immediate short positive (ISP). The immediate operand is treated as a positive integer between 1 and 16.

4.3.7.2 Immediate short negative (ISN). The immediate operand is treated as a negative integer between 1 and 16. Its internal form shall be a 2's complement, sign-extended 16-bit number.

4.3.8 Instruction counter relative (ICR). This addressing mode is used for 16-bit branch instructions. The contents of the instruction counter minus one (i.e., the address of the current instruction) is added to the sign extended 8-bit displacement field of the instruction. The sum points to the memory address to which control may be transferred if a branch is executed. This mode allows addressing within a memory region of 80_{16} to $7F_{16}$ words relative to the address of the current instruction.

4.3.9 Base relative (B). An addressing mode in which the content of an instruction specified base register is added to the 8-bit displacement field of the (16-bit) instruction. The displacement field is taken to be a positive number between 0 and 255. The sum points to the memory address of the required operand. This mode allows addressing within a memory region of 256 words beginning at the address pointed to by the base register.

4.3.10 Base relative-indexed (BX). The sum of the contents of a specified index register and a specified base register is the address of the required operand. Registers R1, R2, ..., R15 may be specified for indexing.

4.3.11 Special (S). The special addressing mode is used where none of the other addressing modes are applicable.

4.4 Registers and support features.

4.4.1 General registers. The instruction set shall support a minimum of 16 registers (R0 through R15). The registers may be used as accumulators, index registers, base registers, temporary operand memory, and stack pointers with the following restrictions:

- a. Only registers R1, R2, ..., R15 may be used as index registers (RX).
- b. Only four registers, R12, R13, R14, and R15 may be used as base registers for instructions having the Base Relative address mode.
- c. R15 is the implicit stack pointer for the Push and Pop Multiple instructions (Opcode $8F_{16}$ and $9F_{16}$).
- d. The general registers are not in the logical memory address space.
- e. Instructions having the Base Relative addressing mode have a single accumulator. The register pair (R0, R1) is the accumulator for double precision and floating point operations. Register R2 is the accumulator for single precision operations except multiply and divide base relative also use R3.

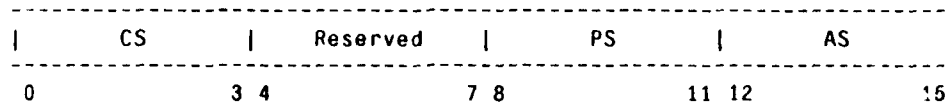
The general registers shall functionally appear to be 16 bits in length. For instructions requiring a 32-bit operation, adjacent registers shall be concatenated to form effective 32-bit registers. Instructions requiring 48-bit operation shall concatenate three adjacent registers to form an effective 48-bit register.

When registers are concatenated, the register specified by the instruction shall represent the most significant word. The register set wraps around, that is, R15 concatenates with R0 for 32-bit operations and R15 concatenates with R0 and R1 for 48-bit operations.

4.4.2 Special registers. The instructions shall make use of the following special registers: instruction counter, status word, fault register, interrupt mask, pending interrupt register, and input/output interrupt code registers.

4.4.2.1 Instruction counter (IC). A 16-bit register used for program sequencing. It allows instructions within a range of 65,536 words to be executed. It is external to the general registers. It is saved in memory when an interrupt is serviced.

4.4.2.2 Status word (SW). The instruction set shall reference a 16-bit status word register whose state is defined by some prior event occurrence in the computer. The figure below indicates the format for the SW with the following paragraphs describing the meaning of the Condition Status (CS) field, reserved bits, the Processor State (PS) field, and the Address State (AS) field.



CS Bits: A four-bit field (bits 0 through 3) of the status word shall be dedicated to instruction results (i.e., instruction status bits) and is defined as condition status (CS). Bits 0, 1, 2, and 3 shall be identified as C, P, Z, and N, respectively, and their meanings are given by the following register transfer description:

$C = (CS)_0 = 1$ if result generates a carry from an addition or no borrow from a subtraction

$P = (CS)_1 = 1$ if result is greater than (zero)

$Z = (CS)_2 = 1$ if result is equal to (zero)

$N = (CS)_3 = 1$ if result is less than (zero)

Reserved Bits: Bits 4 through 7 of the status word shall be reserved.

PS Bits: A four-bit field (bits 8 through 11) of the status word shall be dedicated to the processor state (PS) code. The code value defined by the PS shall be used for the following two functions:

For implementations which include the memory access lock feature of the expanded memory addressing option (see paragraph 4.5.2.2), PS shall define the memory access key code for all instructions and operand references to memory. References to memory during the interrupt recognition sequence for vector table pointer fetches and linkage/service parameter store/read references shall not use PS to define the memory access key code, but shall use an implied $PS=0$ value.

PS shall determine the legal/illegal criteria for privileged instructions. When $PS=0$ and a privileged instruction execution is attempted, the instruction shall be legal and shall be executed properly as defined. When $PS \neq 0$ and a privileged instruction execution is attempted, the

instruction shall be illegal, shall be aborted, and the privileged instruction fault bit in the fault register (FI₁₀) shall be set to one.

AS bits: A four-bit field (bits 12 through 15) of the status word shall be dedicated to the address state (AS) code. For implementations which do not include the expanded memory addressing option, an address state fault shall be generated for any operation which attempts to modify AS to a non-zero value. For implementations which include the expanded memory addressing option, AS shall define the group (pair) of page register sets to be used for all instruction and operand references to memory. References to memory during the interrupt recognition sequence for vector table pointer fetches and service parameter load references shall not use AS to define the operand page register set, but shall use an implied AS = 0 value. The linkage parameter store references shall use the AS field of the new status word. For partial implementations which include less than 16 groups of page register sets for the expanded memory addressing option (see paragraph 4.5.2.3), the address state fault bit in the fault register (FI₁₁) shall be set to one if any operation attempts to establish an AS value that is not implemented.

4.4.2.3 Fault register (FI). The fault register is a 16-bit register used for indicating machine error conditions. The logical OR of the fault register bits is used to generate the machine error interrupt. The fault register shall be read and cleared by an XIO instruction. If a particular fault bit is not implemented, then the bit shall be set to zero. The fault bits shall be assigned as specified in the following:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MEMORY		PARITY		I/O		SPARE		ILLEGAL			RES.		BITE		
PROTECT															

The bits shall have the following meaning when set to one (1):

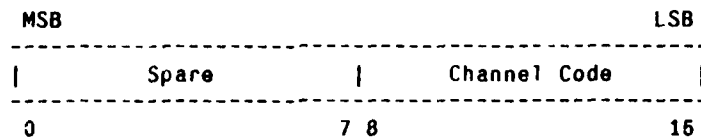
- Bit 0: CPU Memory Protection Fault. The CPU has encountered an access fault, write protect fault, or execute protect fault.
- Bit 1: DMA Memory Protection Fault. A DMA device has encountered an access fault or a write protect fault.
- Bit 2: Memory Parity Fault.
- Bit 3: PIO Channel Parity Fault.
- Bit 4: DMA Channel Parity Fault.
- Bit 5: Illegal I/O Command Fault. An attempt has been made to execute an unimplemented or reserved I/O command.
- Bit 6: PIO Transmission Fault. Other I/O error checking devices, if used, may be ORed into this bit to indicate an error.
- Bit 7: Spare.
- Bit 8: Illegal Address Fault. A memory location has been addressed which is not physically present.
- Bit 9: Illegal Instruction Fault. An attempt has been made to execute a reserved code.

- Bit 10: Privileged Instruction Fault. An attempt has been made to execute a privileged instruction with PSA=0.
- Bit 11: Address State Fault. An attempt has been made to establish an AS value for an unimplemented page register set.
- Bit 12: Reserved.
- Bit 13: Built-in Test Fault. Hardware built-in test equipment (BITE) error has been detected.
- Bit 14-15: Spare BITE. These bits are for use by the designer for future defining (coding, etc.) the BITE error which is detected. This can be used with Bit 13 to give a more complete error description.

4.4.2.4 Interrupt mask (MK). The interrupt mask register is software controlled and contains a mask bit for each of the system interrupts. The interrupt system is defined in paragraph 4.6.

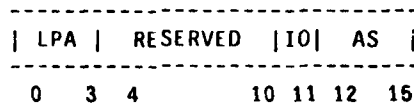
4.4.2.5 Pending interrupt register (PI). The pending interrupt request register is software and hardware controlled and contains the pending interrupts that are attempting to vector the instruction counter. A pending interrupt is set by a system interrupt signal. The pending interrupt bit that generates the interrupt request is cleared by hardware action during the interrupt processing prior to initiating software at the address defined by the new IC value. The register may be set, cleared, and read by the I/O instructions.

4.4.2.6 Input/output interrupt code registers (IOIC)(optional). The input/output interrupt code registers, if implemented, are used to indicate which channel generated the input/output interrupt. One register is assigned for each of the two input/output interrupts. Each register is set by hardware to reflect the address of the highest priority channel requesting that level of interrupt. The address shall be 00_{16} for channel number 0, 01_{16} for channel number 15, $7F_{16}$ for channel number 127, etc. The IOICs shall not be altered once the interrupt sequence has commenced until they are read by an I/O instruction.



4.4.2.7 Page registers (optional). Up to 256 sixteen bit registers for optional expanded memory addressing.

4.4.2.8 Memory fault status register (MFSR)(optional). The memory fault status register provides the page register selection designators associated with memory faults. The page register designators (below) captured by the MFSR are valid for the memory reference causing the fault.



- LPA: Address of page register within the set.
- RESERVED: Must not be used.
- IO: Instruction/operand page set selector (I = instruction).
- AS: Address of selected group.

4.4.3 Stack. The instruction set shall support a stack mechanism. The operation of the stacking mechanism shall be such that the "last-in, first-out" concept is used for adding items to the stack and the Stack Pointer (SP) register always contains the memory address where the last item is stored on the stack. The stack provides for nested subroutine linkage using register 15. The stack shall also reside in a user defined memory area. Two instructions shall use register number 15 (R15) as the implied system stack pointer: Push Multiple registers, PSHM (see page 87), and Pop Multiple registers, POPM (see page 77). The stack expands linearly toward zero as items are added to it.

Two instructions, Stack IC and Jump to Subroutine, SJS (see page 68), and Unstack IC and Return from Subroutine, URS (see page 69), allow the programmer to specify any of the 16 general registers as the stack pointer. The memory block immediately preceding the stack area may be protected (by user using memory protect RAM), thus providing a means of knowing (memory protect interrupt) when the stack limit is exceeded. The stack shall be addressed by the Stack IC and Jump to Subroutine, Unstack IC and Return from Subroutine, Push Multiple, and Pop Multiple instructions.

4.4.4 Processor initialization.

4.4.4.1 Processor reset state. Table VI defines the processor reset state:

TABLE VI. Processor reset state

<u>Register/Device/Function</u>	<u>Condition After Reset</u>
Instruction Counter	All zeros
Status Word	All zeros
Fault Register	All zeros
Pending Interrupt Register	All zeros
Interrupt Mask Register	All zeros
General Registers	Indeterminate
Interrupts	Disabled
Timers A & B	Started and all zeros ¹
Page Registers	Group 0 enabled ¹
Page Registers AL Field	All zeros ¹
Page Registers W Field	Zero ¹
Page Registers E Field	Zero ¹
Page Registers PPA field	Exact logical to physical ¹
Memory Protect RAM	Disabled and all zeros ^{1 2}
Start Up ROM	Enabled ¹
DMA Enable	Disabled ¹
Input Discretes	Indeterminate ¹
Trigger Go Indicator	Started ¹
Discrete Outputs	All zeros ¹

¹ If implemented (optional)

² Main Memory Globally Protected

4.4.4.2 Power up. Upon application of power, the processor shall enter the reset state, the normal power up discrete shall be set (if implemented), and execution shall begin.

4.4.5 Interval timers (optional). If implemented, then two interval timers shall be provided in the computer and shall be referred to as Timer A and Timer B. Both timers can be loaded, stopped, started, and read with the commands described in the NIO paragraph (see page 29). The two timers shall be 16-bit counters which operate as

follows. Effectively, a one is automatically added to the least significant bit of the timer. Bit fifteen is the least significant bit and shall represent the specified increment value of that timer, i.e., either 10 or 100 microseconds. An interrupt request is generated when a timer increments from $FFFF_{16}$ to 0000_{16} . After power up, if the timers are not loaded by software, then an interrupt request is generated after 65,536 counts. A sample of the 16-bit counting sequence (shown in hex) is 0000, 0001, ..., 7FFF, 8000, ..., FFFF, 0000, ... At system reset or power up, the timers are initialized in accordance with paragraph 4.4.4.1. The timers are halted when a breakpoint, BPT (see page 138), instruction is executed and the console is connected.

4.5 Memory.

4.5.1 Memory addressing. The instruction set shall use 16-bit logical addresses to provide for referencing of 65,536 words. When the expanded memory option (see paragraph 4.5.2) is not implemented, physical addresses shall equal logical addresses.

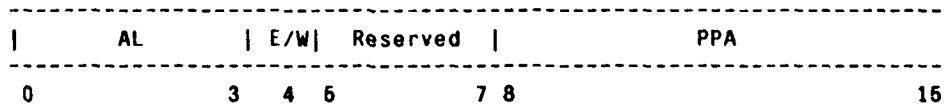
4.5.1.1 Memory addressing arithmetic. Arithmetic performed on memory logical addresses shall be modulo 65,536 such that references to the maximum logical address of $FFFF_{16}$ plus 1 shall be to logical address 0000_{16} .

4.5.1.2 Memory addressing boundary constraints. There shall be no odd or even memory address boundary constraints.

4.5.2 Expanded memory addressing (optional). If used, then expanded memory addressing shall be performed via a memory paging scheme as depicted in figure 1. There shall be a maximum of 512 page registers in the page file (not in logical memory space). These shall functionally be partitioned into 16 groups with 2 sets per group and 16 page registers per set. Within a group, one set shall be designated for instruction references and the other set for operand references. The page size shall be 4096 words such that one set of 16 page registers shall be capable of mapping 65,536 words defined by a 16-bit logical address. The page group shall be selected by the 4-bit Address State (AS) field of the Status Word (SW). The instruction/operand set within the group shall be selected by the hardware that differentiates between instruction and operand memory references. The 4 most significant bits of any 16-bit logical address shall select the page register within that set. The 8-bit Physical Page Address (PPA) within the page register shall be concatenated with the 12 least significant bits of the logical address to form a 20-bit physical address, allowing addressing of 1,048,576 words of physical memory.

4.5.2.1 Group selection. During instruction and operand references to memory, the address state (AS) field of the status word shall be used to designate the page file group. During an interrupt recognition sequence, the operand set of group zero shall be used for vector table and pointer references to memory.

4.5.2.2 Page register word format. Each page register shall be 16 bits. The figure below indicates the format for the page register words with the following paragraphs describing the meaning of the access lock (AL) field, the execute protect (E) bit, the write protect (W) bit, reserved bits, and the Physical Page Address (PPA) field.



AL Field: The access lock and key feature is optional if expanded memory addressing is implemented. If the access lock and key feature is not implemented, then the AL field shall always be zero. If it is implemented, then a 4-bit field (bits 0 through 3) of each page register shall contain the access lock (AL) code for the associated page register, which shall be used with the access key codes to determine access permission. The access key codes may be supplied by either the status word or the DMA channel. For each of the possible 16 values of the AL code, access shall be permitted for the reference according to table VII. References supplying an unacceptable access key code shall not modify any memory location or general registers and an access fault shall be generated. An access fault resulting from a CPU reference attempt shall set fault register bit 0 to cause a machine error interrupt. An access fault

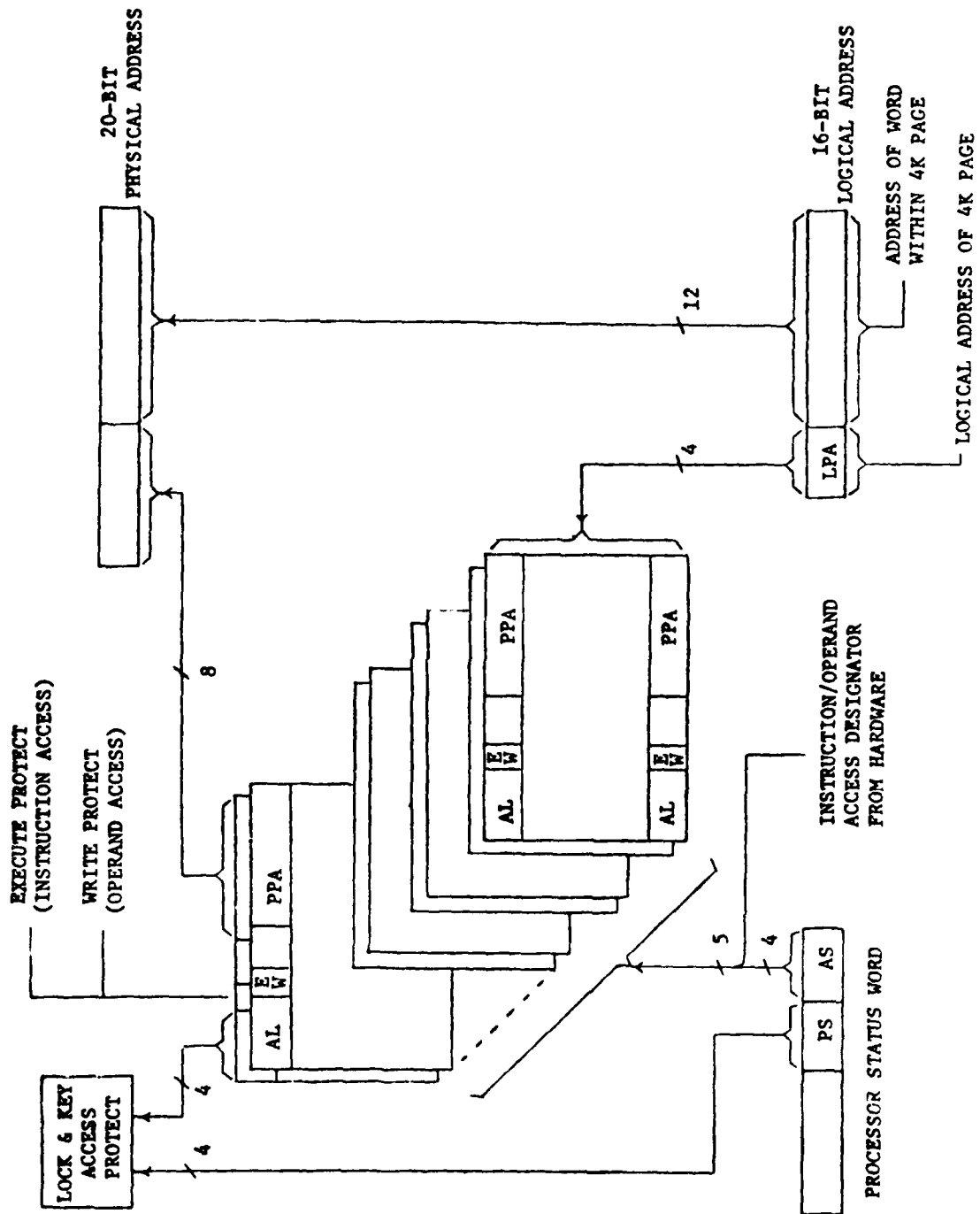


FIGURE 1. Expanded memory mapping diagram

TABLE VII. AL code to access key mapping

<u>AL Code</u>	<u>Acceptable Access Key Codes</u>
0	0
1	0, 1
2	0, 2
3	0, 3
4	0, 4
5	0, 5
6	0, 6
7	0, 7
8	0, 8
9	0, 9
A	0, A
B	0, B
C	0, C
D	0, D
E	0, E
F	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

resulting from a DMA attempt shall set fault register bit 1 to cause a machine error interrupt. Note that the access lock and key codes defined in the above table have the following characteristics:

- An access lock code of F_{16} is an "unlocked" lock code and allows any and all access key codes to be acceptable.
- An access key code of 0 is a "master" key code and is acceptable to any and all access lock codes.
- Access key codes 1 through F_{16} are acceptable to only their own "matched" lock code or the "unlocked" lock code of F_{16} .
- An access key code of F_{16} is acceptable to only the "unlocked" lock code of F_{16} .

E Bit: For instruction page register sets only, bit 4 shall be defined as the E bit and shall determine the acceptable/unacceptable criteria for read references for instruction fetches. When $E=1$, any attempted instruction read reference designating that associated page register shall be terminated and an execute protect fault shall be generated. An execute protect fault shall set fault register bit 0 to cause a machine error interrupt.

W Bit: For operand page registers only, bit 4 shall be defined as the W bit and shall determine the acceptable/unacceptable criteria for write references. When $W=1$, any attempted write reference designating that associated page register shall not modify any memory location and a write protect fault shall be generated. A write protect fault resulting from a CPU reference attempt shall set fault register bit 0 to cause a machine error interrupt. A write protect fault resulting from a DMA reference attempt shall set fault register bit 1 to cause a machine error interrupt.

Reserved Bits: Bits 5 through 7 of all of the page registers shall be reserved and shall always be 0.

MIL-STD-1750A (USAF)

2 July 1980

PPA Field: An eight-bit field (bits 8 through 15) of each page register shall be dedicated to the physical page address which is used to define the physical address as depicted in figure 1.

4.5.2.3 Partial implementations of expanded memory addressing. A given implementation of this standard may include a partial implementation of the expanded addressing option. That partial implementation may use 2, 4, or 8 groups of page registers as follows:

<u>Number of Groups</u>	<u>AS Group Codes</u>
2	0 and 1
4	0 through 3
8	0 through 7

Within any full or partial implementation, the lock feature may or may not be included.

4.5.3 Memory parity (optional). If used, then bit 2 in the fault register shall be set to indicate a memory parity error.

4.5.4 Memory block protect (optional). If used, shall be as described by the input/output instructions. For operations which contain multiple memory references, each store operation shall be as defined by the memory protection for that specific memory address.

4.5.5 References to unimplemented memory. Attempted access to physical addresses which are not implemented shall generate an illegal address fault and shall cause the referencing action to terminate. An illegal address fault shall set fault register bit 8 to cause a machine error interrupt.

4.5.6 Start up ROM (optional). If used, the start up read only memory (ROM) address range shall be contiguous starting from address 0 up to a maximum of 65,536, as required by the system application. When the start up ROM is enabled, if an I/O or CPU store function is executed whose address is within the start up ROM, then the store is attempted into the main memory. When the start up ROM is enabled, if a read function (instruction or operand) is executed from either I/O or the CPU whose address is to the start up ROM, then the read shall be from the start up ROM. When disabled, the start up ROM cannot be accessed.

4.5.7 Reserved memory locations. Locations 2 through $1F_{16}$ are reserved. Locations 20_{16} through $3F_{16}$ are used by the hardware and the stored program as defined by table VIII.

4.6 Interrupt control.

4.6.1 Interrupts. The instruction set shall support a minimum of sixteen (16) interrupts as shown in table VIII. An interrupt request may occur at any time; however, the interrupt processing must wait until the current instruction is completed. An exception to this is the Move Multiple Word which may be interrupted after each single word transfer. The overall procedure for acceptance of, responding to, and processing of an interrupt shall be as illustrated by the flow chart of figure 2.

4.6.1.1 Interrupt acceptance. The interrupt system shall have the capability to accept external and internal interrupts. Figure 2 indicates the relationship between the interrupt signals, the pending interrupt register, the interrupt mask register, the priority control logic, the software controllable/accessible signals and the fundamental communications between the interrupt system and the CPU.

4.6.1.2 Interrupt software control. Software shall be able to input from or output to the interrupt mask register as well as the pending interrupt register. Also, software shall be able to disallow recognition of interrupts via the "disable interrupt" signal (without inhibiting interrupt acceptance into the pending interrupt register) and to allow recognition of interrupts via the "enable interrupts" signal. The disabling shall not allow any interrupts after the beginning of the disable instruction. The CPU's interrupt service hardware shall continue to "disable interrupts" for one instruction after the enable interrupts instruction is completed. Full descriptions of interrupt instructions

MIL-STD-1750A (USAF)
2 July 1980

TABLE VIII. Interrupt definitions

Interrupt Number	Interrupt Mask Bit Number	Interrupt Linkage Pointer Address (Hex)	Interrupt Service Pointer Address (Hex)	
0	0	20	21	Power Down (cannot be masked or disabled)
1	1	22	23	Machine Error (cannot be disabled)
2	2	24	25	Spare
3	3	26	27	Floating Point Overflow
4	4	28	29	Fixed Point Overflow
5	5	2A	2B	Executive Call (cannot be masked or disabled)
6	6	2C	2D	Floating Point Underflow
7	7	2E	2F	Timer A (if implemented)
8	8	30	31	Spare
9	9	32	33	Timer B (if implemented)
10	10	34	35	Spare
11	11	36	37	Spare
12	12	38	39	Input/Output Level 1 (if implemented)
13	13	3A	3B	Spare
14	14	3C	3D	Input/Output Level 2 (if implemented)
15	15	3E	3F	Spare

Notes: Interrupt number 0 has the highest priority. Priority decreases with increasing interrupt number.

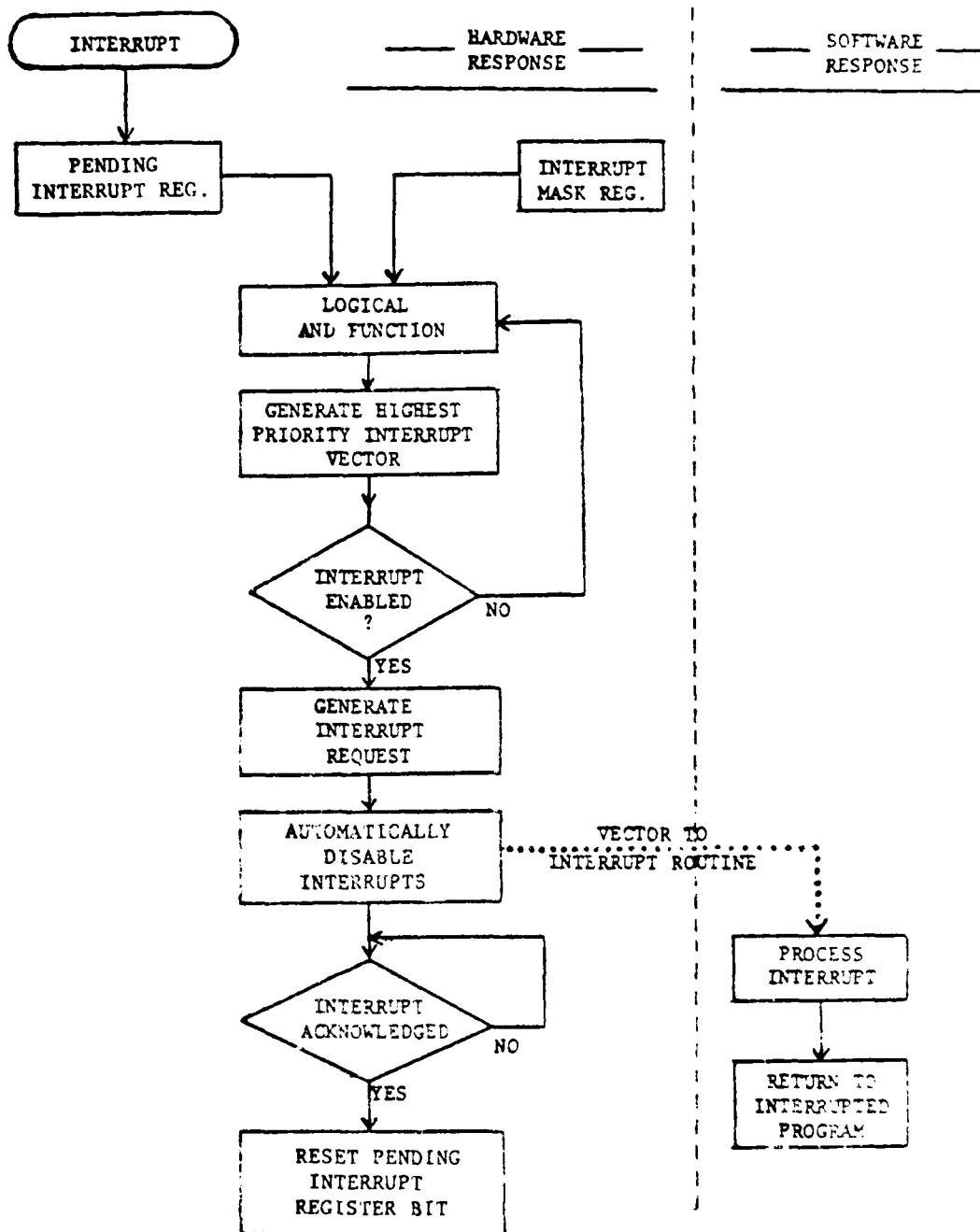


FIGURE 2 Interrupt system flowchart

are given in the input/output instruction repertoire.

4.6.1.3 Interrupt priority definitions. The priority definitions of the interrupts and their required relationship to the interrupt mask and interrupt pointer addresses are illustrated in table VIII, Interrupt Definitions. The power down interrupt shall initiate the power down sequence and cannot be masked or disabled during normal operation of the computer. The executive call interrupt, used with the Branch to Executive instruction, BEX, (see page 62) also cannot be masked or disabled. The machine error interrupt cannot be disabled but can be masked during normal operation of the computer. All other interrupts can be disabled and masked. If a floating point overflow/underflow or fixed point overflow condition occurs, then the instruction generating that condition shall be interrupted at its completion if the interrupt is unmasked and enabled.

4.6.1.4 Interrupt vectoring mechanism. The vectoring mechanism shall be as illustrated on figure 3. For each interrupt there shall be two fixed memory locations in the "vector table": (1) the first memory location (Linkage Pointer) shall be defined as the address of where to store the current (old) state of the computer (i.e., "old interrupt mask", "old status word", and "old instruction counter"); and (2) the second memory location (Service Pointer) shall be defined as the address of the next (new) state of the computer (i.e., "new interrupt mask", "new status word", and "new instruction counter"). Returning from interrupts may be accomplished by executing the Load Status (LSI/LSII) instruction with the value/address of the Linkage Pointer for an address. field.

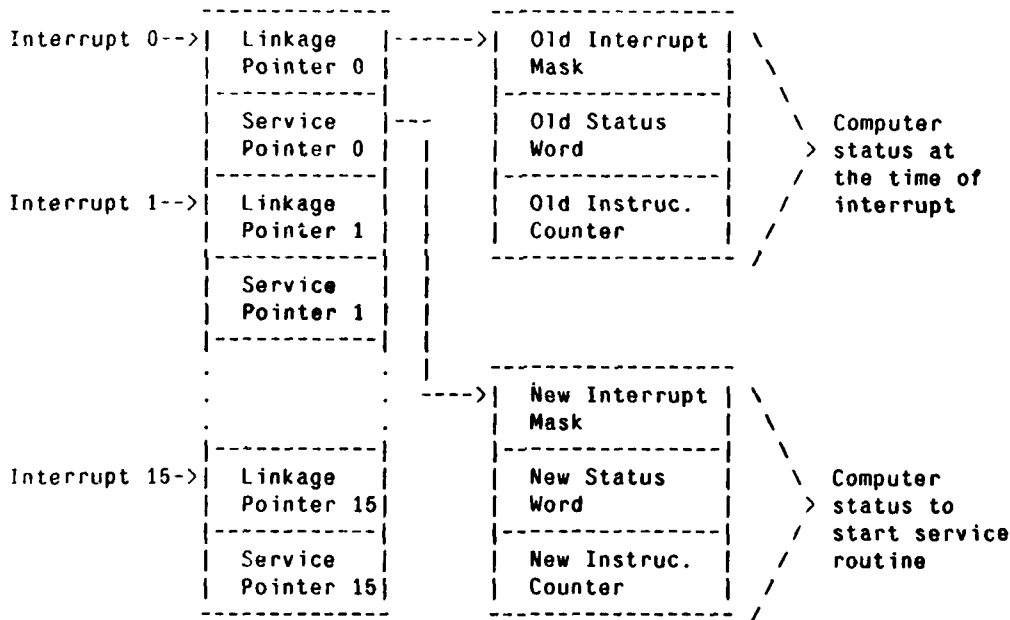


FIGURE 3. Interrupt vectoring system

4.7 Input/output. In conjunction with the spare command codes, the I/O interrupts, and the I/O interrupt code registers, the I/O instructions provide a framework within which the user can implement his system interfaces. The particulars of the system interfaces outside of this framework (such as dedicated memory locations, channel register definitions, command code assignments/definitions, multiple channel priorities, page register access, etc.) are not included in this standard.

2 July 1980

4.7.1 Input. The input instructions transfer data from an external I/O device or an internal special register to a CPU general register. This command is used to read data from peripheral devices, timers, status word, fault register, discrettes, interrupt mask, etc. A full description of the input instructions is given in the instruction repertoire.

4.7.2 Output. The output instructions transfer data from a CPU general register to an external I/O device or special register. This command is used to write data to peripheral devices, discrettes, start and stop timers, enable and disable interrupts and DMA, set and clear interrupt requests, masks and pending interrupt bits, etc. A full description of the output instructions is given in the instruction repertoire.

4.7.3 Input/output commands. Input/output commands are classified as mandatory, optional, reserved, or spare. Mandatory I/O commands must be implemented as defined. Optional I/O commands must be implemented as defined, if implemented. Reserved I/O commands must not be implemented. Spare I/O commands may be implemented as required by the application. Attempted execution of an unimplemented optional or spare I/O command or a reserved I/O command shall cause the illegal I/O command fault to be set in the fault register (FT₅) causing a machine error interrupt. Input/output command words shall be fully decoded. "TBDS" in input/output instruction descriptions refer to parameters to be determined by the application system requirements. Within these classifications, the use of the command is defined in the instruction description.

4.7.4 Input/output command partitioning. The I/O command space shall be divided into 128 channels. Up to 512 commands within each channel group (256 input and 256 output) may be used with each I/O interface. Table IX lists the 128 I/O channel groups. The attempted execution of an unimplemented I/O command shall cause bit 5 of the fault register to be set, generate a machine error interrupt, and abort to completion.

4.7.5 Input/output interrupts (optional). Input/output level 1 and level 2 interrupts are available to the user. Either interrupt level or both may be implemented for an interface as defined by the particular application specification. The interrupts shall be used in conjunction with the input/output interrupt code registers to provide I/O channel to process communications. Two levels of interrupts allow easy differentiation of normal reporting from error reporting.

4.7.6 Dedicated I/O memory locations. If dedicated memory locations are used to communicate information to and/or from an I/O channel, these locations shall be consecutive memory locations starting at an implementation defined location. Locations 40₁₆ through 4F₁₆ are optional for I/O usage.

4.8 Instructions.

4.8.1 Invalid instructions. Attempted execution of an instruction whose first 16 bits are not defined by this standard shall cause the invalid instruction bit in the fault register (FT₉) to be set generating a machine error interrupt. All undefined bit patterns in the first 16 bits of an instruction are reserved.

4.8.2 Mnemonic conventions. Each instruction has an associated mnemonic convention. In general, the operation is one or two letters, e.g., L for load, A for add, ST for store.

Floating point operations have a prefix of F, e.g., FL for floating load, FA for floating add.

Double precision operations have a prefix of D, e.g., DL for double load, DA for double add.

Extended precision floating point operations have a prefix of EF, e.g., EFA for extended precision floating point add.

Register-to-register operations have a suffix of R, e.g., AR for single precision add register-to-register, FAR for floating add register-to-register.

Indirect memory reference is indicated by a suffix I, e.g., LI for load indirect.

Immediate addressing, using the address field as an operand, is indicated by a suffix of IM, e.g., AIM for single

TABLE IX. Input/output channel groups

<u>Output</u>	<u>Input</u>	<u>Usage</u>
00XX	80XX \	PIO
03XX	83XX /	
04XX	84XX \	Spare
1FXX	9FXX /	
20XX	A0XX	Processor & Auxiliary Register Control
21XX	A1XX \	Reserved
2FXX	AFXX /	
30XX	B0XX \	Spare
3FXX	BFXX /	
40XX	C0XX	Processor & Auxiliary Register Control
41XX	C1XX \	Reserved
4FXX	CFXX /	
50XX	D0XX	Memory Protect RAM
51XX	D1XX \	Memory Address Extension (page register commands)
52XX	D2XX /	
53XX	D3XX \	Spare
7FXX	FFXX /	

precision add immediate.

Use of indexing is specified in assembly language by the occurrence of the operational field after the address field.

e.g., FA A2,A1.PHA,A5: floating add to register A2 from memory location ALPHA indexed by register A5.

4.8.3 Instruction matrix. Table X contains the order type matrix which relates each instruction operation code to an assigned symbol. The numbers shown across the top of the matrix are hexadecimal numbers which represent the higher order four bits of the operation code, and the hexadecimal numbers along the left side represent the lower order four bits of the operation code. Table XI contains the order types and assigned mnemonics for the extended Operation Code instructions.

4.8.4 Instruction set notation. The text and register transfer descriptions are intended to complement each other. Ambiguities or omissions in one are resolved by the other. The following definitions and special symbols are associated with the instruction descriptions.

MIL-STD-1750A (USAF)

2 July 1983

CPU Registers

R0, R1, ..., R15	The 16, 16-bit general registers
IC	Instruction Counter
SW	Status Word
CS	<i>Condition Status.</i> A 4 bit quantity that is set according to the result of instruction executions.
IP	Linkage Pointer
SP	Stack Pointer; R15 for the Push and Pop Multiple instructions
SVP	Service Pointer
MK	Interrupt Mask Register
PI	Pending Interrupt Register
RA, RB	An unspecified general register

Addressing Modes

R	Register Direct
D, DX	Memory Direct, Memory Direct-Indexed
I, IX	Memory Indirect, Memory Indirect with Pre-Indexing
IM, IMX	Immediate Long, Immediate Long with Indexing
ISP, ISN	Immediate Short with Positive Operand, Immediate Short with Negative Operand
ICR	IC-Relative
B, BX	Base Relative, Base Relative with Indexing
S	Special

Data Quantities

MSH, LSH	Most Significant Half, Least Significant Half
MSB, LSB	Most Significant Bit, Least Significant Bit
S.P., D.P., Ft. P., E.F.P.	Abbreviation for "Single Precision," "Double Precision," "Floating Point," and "Extended Floating Point" operations, respectively.
MO	Floating Point Derived Operand mantissa (fractional part): DO _{0,23} (Ft.P), DO _{0,23} DO _{32,47} (E.F.P.)

EO	Floating point 8-bit 2's complement Derived Operand characteristic (exponent): $1X_{24-31}$ MA
	Floating point register accumulator mantissa (fractional part): $(RA, RA + 1)_{0-23} (1 + P)$, $(RA, RA + 1)_{0-23} (RA + 2)_{32-47} (1 + P)$
EA	Floating point 8-bit 2's complement register accumulator characteristic (exponent): $(RA, RA + 1)_{24-31}$
RQ, MP, MQ	An entity used for register level transfer description clarification. These registers are not part of the general register file.

Miscellaneous

(X)	Contents of Register X
(X, X + 1)	Contents of concatenated Registers X and X + 1
[X]	Contents of memory address X
[X, X + 1]	Contents of sequential memory locations X and X + 1
OVM	Mantissa (fractional part) overflow
Exit	Indicates termination of present register transfer operation (except the setting of the CS bits)
DA	Derived Address
DO	Derived Operand
N, M, n	An integer number
DSPL	Displacement
X_n	If X is a CPU register or a data quantity (see above), then n specifies a bit position in X. If X is not a CPU register or a data quantity, then the number X is to the base n. If X is a number and $n = 16$, then X is a 2's complement hexadecimal number.
X^i	If X is a CPU register or a memory address, then i specifies the state of X. This notation is used in the register transfer descriptions to refer to the contents of a CPU register or a memory address at different times (states) of the execution of the instruction. If X is not a CPU register or a memory address, then the number X is raised to the ith power.

Symbols

$\leftarrow$	Unilateral transfer designator
$\leftrightarrow$	Bilateral transfer designator
$\sim$	Comparison Designator
x	Indicates a "don't care" bit when used in a binary number

MIL-STD-1750A (USAF)
2 July 1980

$>$ Greater than
 $<$

Less than

$=$ Equals

$\geq$ Greater than or equal

$\leq$ Less than or equal

$\uparrow$ Logical AND

$\vee$ Logical OR

$\oplus$ Exclusive OR

$-$ Logical NOT

$\parallel$ Absolute value

TABLE X. Operation code matrix

LOAD STORE	INTEGER ARITHMETIC	FIXING POINT	LOGICAL COMPARE	OPCODE EXTENSIONS	BIT SHIFT	JUMP	LOAD	STORE	ADD	SUB	MULT	DIVIDE	LOGICAL COMPARE		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 LB BP12	AR BP12	FAR BP12	OPR BP12	BRZ BR12*	SR	SLL	JC	L	ST	A	S	MS	DV	OR	C
1 LB BP13	AR BP13	FAR BP13	OPR BP13	BRZ BR13*	SBR	SRL	JCI	LR	STC	AR	SR	MSR	DVR	ORP	CB
2 LB BP14	AR BP14	FAR BP14	OPR BP14	BRZ BR14*	SRI	SRA	JS	LISP	STCI	AISP	STSP	MSLP	DISP	AUD	CISP
3 LB BP15	AR BP15	FAR BP15	OPR BP15	BRZ BR15*	RB	SIC	SOJ	LISN	MOV	INCM	DECM	MISM	DISM	ANDP	CISM
4 DB BP12	AR BP12	FSB BP12	ANDR BP12		RBR		FR	LT	STL	ARS	NEG	M	D	PCR	CHL
5 DB BP13	AR BP13	FSB BP13	ANDR BP13		RBI	DSLL	DEZ	LIM		DADS	DMEG	MR	DM	FOR	
6 DB BP14	AR BP14	FSB BP14	ANDR BP14		TR	DSRL	RLT	OL	OST	DA	DS	DM	DO	M	DC
7 DB BP15	AR BP15	FSB BP15	ANDR BP15		TBR	DSRA	BER	DLR	SPM	DAR	DSR	DAR	DOR	NR	DCR
8 SIB BP12	PB BP12	FMB BP12	CB BP12	XIO*	TBI	OSLC	RLF	DLI	OSTI	FA	FS	FM	FO	FLX	FC
9 SIB BP13	PB BP13	FMB BP13	CB BP13	XIO*	TBI	OSLC	RLF	LM	SIM	FAR	FSR	FMR	FOR	FLT	FCP
A SIB BP14	PB BP14	FMB BP14	CB BP14	IMPL*	SVBR	SIR	BRZ	EFL	FFST	EFA	EFF	EFM	EFC	EFIX	EFC
B SIB BP15	PB BP15	FMB BP15	CB BP15		RVR	SAR	RGE	LUB	STUB	EFAR	EFAR	EFMR	EFOR	EFLY	EFCR
C DSB BP12	DB BP12	FDB BP12	FCB BP12		RVR	SCR	LSTI*	LLR	STLB	FARS	FNEG			ZBP	
D DSB BP13	DB BP13	FDB BP13	FCB BP13			OSLP	LSTI*	LUBI	SUBI					ZBP	
E DSB BP14	DB BP14	FDB BP14	FCB BP14		TVBR	DSAR	SJS	LLBI	SLBI						
F DSB BP15	DB BP15	FDB BP15	FCB BP15			OSCR	URS	POPM	PSHM						NOP/BPT

* These order types represent instructions which have "extended" operation codes and are fully described in the instruction specifications and in Table 4-5.

Privileged instructions

TABLE XI. Extended operation codes

Opcode Extension (see below for location)

PSN	Format	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	BRA	LBR	OLBR	STBR	OSTR	ABR	SBBX	MBX	DBX	FABX	FSBR	FMBX	FDBX	CBA	FCBA	AMDX	ORBX
41	BRA	LBR	OLBR	STBR	OSTR	ABR	SBBX	MBX	DBX	FABX	FSBR	FMBX	FDBX	CBA	FCBA	AMDX	ORBX
42	BRA	LBR	OLBR	STBR	OSTR	ABR	SBBX	MBX	DBX	FABX	FSBR	FMBX	FDBX	CBA	FCBA	AMDX	ORBX
43	BRA	LBR	OLBR	STBR	OSTR	ABR	SBBX	MBX	DBX	FABX	FSBR	FMBX	FDBX	CBA	FCBA	AMDX	ORBX
44	IMUL		AIM	SIM	MIM	MSIM	DIM	DVIM	ANDM	ORIM	FORM	CIM	NIM				

*MSH (Most Significant Half)

MSH

**Base Relative Indexed Format

Op Code | BR | Op. Ex. | RX |

**Immediate Opcode Extension Format

Op Code | RA | Op. Ex. | Operand |

5 DETAILED REQUIREMENTS

5.1 Execute input/output.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4 16
IM	XIO	RA, CMD	-----
IMX	XIO	RA, CMD, RX	48 RA RX CMD

DESCRIPTION: The input/output instruction transfers data between an external/internal device and the register RA. The Derived Operand, DO, specifies the operation to be performed or the device to be addressed. The immediate operand field may be viewed as an operation code extension field. Note that if indexing is specified, then the input/output operation or device address is formed by summing the contents of the register RX and the immediate field. This is a privileged instruction.

The mandatory and optional input/output immediate command fields are listed below.

Mandatory XIO Command Fields and Mnemonics

0YXX PO	Programmed Output: This command outputs 16 bits of data from RA to a programmed I/O port. Y may be from 0 through 3.
2000 SMK	Set Interrupt Mask: This command outputs the 16-bit contents of the register RA to the interrupt mask register. A "1" in the corresponding bit position allows the interrupt to occur and a "0" prevents the interrupt from occurring except for those interrupts that are defined such that they cannot be masked.
2001 CLR	Clear Interrupt Request: All interrupts are cleared (i.e., the pending interrupt register is cleared to all zeros) and the contents of the fault register are reset to zero.
2002 ENBL	Enable Interrupts: This command enables all interrupts which are not masked out. The enable operation takes place after execution of the next instruction.
2003 DSBL	Disable Interrupts: This command disables all interrupts (except those that are defined such that they cannot be disabled) at the beginning of the execution of the DSBL instruction.
2004 RPI	Reset Pending Interrupt: The individual interrupt bit to be reset shall be designated in register RA as a right justified four bit code. (0 ₁₆ represents interrupt number 0, F ₁₆ represents interrupt number 15). If interrupt I ₁₆ is to be cleared, then the contents of the fault register shall also be set to zero.
2005 SPI	Set Pending Interrupt Register: This command outputs the 16-bit contents of RA to the pending interrupt register. If there is a one in the corresponding bit position of the interrupt mask (same bit set in both the PI and the MK), and the interrupts are enabled, then an interrupt shall occur after execution of the next instruction. If PI ₅ is set to 1, then N is assumed to be 0 (see paragraph 5.30).
200F WSW	Write Status Word: This command transfers the contents of RA to the status word.
8YXX PI	Programmed Input: This command inputs 16 bits of data into RA from the programmed I/O

port. Y may be from 0 through 3.

- A000 RMK Read Interrupt Mask: The current interrupt mask is transferred into register RA. The interrupt mask is not altered.
- A004 RPIR Read Pending Interrupt Register: This command transfers the contents of the pending interrupt register into RA. The pending interrupt register is not altered.
- A00F RSW Read Status Word: This command transfers the 16-bit status word into register RA. The status word remains unchanged.
- A00F RCFR Read and Clear Fault Register: This command inputs the 16-bit fault register to register RA. The contents of the fault register are reset to zero. Bit 1 in the pending interrupt register is reset to zero.

Optional XIO Command Fields and Mnemonics

- 2008 OD Output Discretes: This command outputs the 16-bit contents of the register RA to the discrete output buffer. A "1" indicates an "on" condition and a "0" indicates an "off" condition.
- 200A RNS Reset Normal Power Up Discrete: This command resets the normal power up discrete bit.
- 4000 CO Console Output: The 16-bit contents (2 bytes) of register RA are output to the console. The eight most significant bits (byte) are sent first. If no console is present, then this command is treated as a NOP (see page 137).
- 4001 CLC Clear Console: This command clears the console interface.
- 4003 MPEN Memory Protect Enable: This command allows the memory protect RAM to control memory protection.
- 4004 ESUR Enable Start Up ROM: This command enables the start up ROM (i.e., the ROM overlays main memory).
- 4005 DSUR Disable Start Up ROM: This command disables the start up ROM.
- 4006 DMAE Direct Memory Access Enable: This command enables direct memory access (DMA).
- 4007 DMAID Direct Memory Access Disable: This command disables DMA.
- 4008 TAS Timer A, Start: This command starts timer A from its current state. The timer is incremented every 10 microseconds.
- 4009 TAH Timer A, Halt: This command halts timer A in its current state.
- 400A OTA Output Timer A: The contents of register RA are loaded (i.e., jam transferred) into timer A and the timer automatically starts operation by incrementing from the loaded timer in steps of ten microseconds. Bit fifteen is the least significant bit and shall represent ten microseconds.
- 400B GO Trigger Go Indicator: This command restarts a counter which is connected to a discrete output. The period of time from restart to time-out shall be determined by the system requirements. When the Go timer is started, the discrete output shall go high and remain high for

110 milliseconds, at which time the output shall go low unless another GO is executed. The Go discrete output signal may be used as a software fault indicator.

- 400C TBS Timer B, Start: This command starts timer B from its current state. The timer is incremented every 100 microseconds.
- 400D TBH Timer B, Halt: This command halts timer B at its current state.
- 400E OTB Output Timer B: The contents of register RA are loaded (i.e., jam transfered) into timer B and the timer automatically starts operation by incrementing from the loaded timer in steps of one hundred microseconds. Bit fifteen is the least significant bit and shall represent one hundred microseconds.
- 50XX LMP Load Memory Protect RAM (5000 + RAM address): This command outputs the 16-bit contents of register RA to the memory protect RAM. A "1" in a bit provides write protection and a "0" in a bit permits writing to the corresponding 1024 word memory block. The RAM word MSB (bit 0) represents the lowest number block and the RAM word LSB (bit 15) represents the highest block (i.e., bit 0 represents locations 0 through 1023 and bit 15 represents locations 15360 through 16383 for word zero). Each word represents consecutive 16K blocks of memory. The RAM words of 0 through 63 apply to processor write protect and words 64 through 127 apply to DMA write protect.
- 51XY WIPR Write Instruction Page Register: This command transfers the contents of register RA to page register Y of the instruction set group X.
- 52XY WOPR Write Operand Page Register: This command transfers the contents of register RA to page register Y of the operand set of group X.
- A001 RIC1 Read Input/Output Interrupt Code, Level 1: This command inputs the contents of the level 1 IOIC register into register RA. The channel number is right justified.
- A002 RIC2 Read Input/Output Interrupt Code, Level 2: This command inputs the contents of the level 2 IOIC register into register RA. The channel number is right justified.
- A008 RDOR Read Discrete Output Register: This command inputs the 16-bit discrete output buffer into register RA.
- A009 RDI Read Discrete Input: This command inputs the 16-bit discrete input word into register RA. A "1" indicates an "on" condition and a "0" indicates an "off" condition.
- A00B TPIO Test Programmed Output: This command inputs the 16-bit contents of the programmed output buffer into register RA. This command may be used to test the PIO channel by means of a wrap around test.
- A00D RMFS Read Memory Fault Status: This command transfers the 16-bit contents of the memory fault status register to RA. The fields within the memory fault status register shall delineate memory related fault types and shall provide the page register designators associated with the designated fault.
- C000 CI Console Input: This command inputs the 16-bits (2 bytes) from the console into register RA. The eight most significant bits of RA shall represent the first byte.

MIL-STD-1750A (USAF)
2 July 1980

C001 RCS	Read Console Status: This command inputs the console interface status into register RA. The status is right justified.
C00A ITA	Input Timer A: This command inputs the 16-bit contents of timer A into register RA. Bit fifteen is the least significant bit and represents a time increment of ten microseconds.
C00E ITB	Input Timer B: This command inputs the 16-bit contents of timer B into register RA. Bit fifteen is the least significant bit and represents a time increment of one hundred microseconds.
D0XX RMP	Read Memory Protect RAM (D000 + RAM address): This command inputs the appropriate memory protect word into register RA. A "1" in a bit provides write protection and a "0" in a bit permits writing to the corresponding 1024 word memory block. The RAM word MSB (bit 0) represents the lowest number block and the RAM word LSB (bit 15) represents the highest block (i.e., bit 0 represents locations 0 through 1023 and bit 15 represents locations 15360 through 16383 for word zero). Each word represents consecutive 16K blocks of memory. The RAM words of 0 through 63 apply to processor write protect and words 64 through 127 apply to DMA write protect.
D1XY RIPR	Read Instruction Page Register: This command transfers the 16-bit contents of the page register Y of the instruction set of group X to register RA.
D2XY ROPR	Read Operand Page Register: This command transfer the 16-bit contents of page register Y of the operand set of group X to register RA.
**** *	User defined XIO functions (see table IX).

REGISTER TRANSFER DESCRIPTION: Varies depending on the command field.

REGISTERS AFFECTED: Varies depending on the command field.

5.2 Vectored input/output.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4 16
D	VIO	RA, ADDR	-----
DX	VIO	RA, ADDR, RX	49 RA RX ADDR

DESCRIPTION: The vectored input/output instruction performs the I/O operation as specified by the input/output vector table starting at the derived address, DA, as shown below:

DA	CMD	
DA+1	Vector Select	
DA+2	Data	} one data word for each bit set in the vector select
...	...	

The input/output operation or device address is specified by the sum of the CMD and the product of the bit number of the bit set in the vector select times the contents of RA. This device address is then interpreted as specified by the XIO instruction (see paragraph 5.1) with the exception that I/O data is transferred to or from DA + 2 + i rather than RA (where i starts at zero and is incremented after each transfer). This is a privileged instruction.

REGISTER TRANSFER DESCRIPTION:

- Step 1. $n \leftarrow 0$ and $i \leftarrow 0$;
- Step 2. if $[DA+1]_n = 1$, then I/O command = $[DA] + \{n \times (RA)\}$;
- Step 3. if $[DA+1]_n = 1$, then I/O data = $[DA+2+i]$;
- Step 4. if $[DA+1]_n = 1$, then $i \leftarrow i+1$;
- Step 5. $n \leftarrow n + 1$, exit, if $n = 16$;
- Step 6. go to step 2;

REGISTERS AFFECTED: None

MIL-STD-1750A (USAF)
2 July 1980

5.3 Set bit.

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	
R		SBR N, RB	51	N	RB	
D		SB N, ADDR	8	4	4	, 16
DX	SB	N, ADDR, RX	50	N	RX	ADDR
I		SBI N, ADDR	8	4	4	16
IX	SBI	N, ADDR, RX	52	N	RX	ADDR

DESCRIPTION: Bit number N of the Derived Operand, DO, is set to one. The MSB is designated bit number zero and the LSB is designated bit number fifteen.

REGISTER TRANSFER DESCRIPTION:

$DO_N \leftarrow 1$;

REGISTERS AFFECTED: RB

5.4 Reset bit.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4 ----- 54 N RB -----
R		RBR N, RB	
			8 4 4 16 ----- 53 N RX ADDR -----
D		RB N, ADDR	
DX	RB	N, ADDR, RX	
			8 4 4 16 ----- 55 N RX ADDR -----
I		RBI N, ADDR	
IX	RBI	N, ADDR, RX	

DESCRIPTION: Bit number N of the Derived Operand, DO, is set to zero. The MSB is designated bit number zero and the LSB is designated bit number fifteen.

REGISTER TRANSFER DESCRIPTION:

DO_N <-- 0;

REGISTERS AFFECTED: RB

MIL-STD-1750A (USAF)
2 July 1980

5.5 Test bit.

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	
R		TBR N, RB	57	N	RB	
D		TB N, ADDR	8	4	4	16
DX		TB N, ADDR, RX	56	N	RX	ADDR
I		TBI N, ADDR	8	4	4	16
IX		TBI N, ADDR, RX	58	N	RX	ADDR

DESCRIPTION: Bit number N ($0 \leq N \leq 15$) of the Derived Operand, DO, is tested. Then the Condition Status, CS, is set to indicate non-zero if bit number N of the DO contains a one. Otherwise CS is set to indicate zero. The MSB of the DO is designated bit number zero and the LSB of the DO is designated bit number fifteen.

REGISTER TRANSFER DESCRIPTION:

(CS) <-- 0010 if $DO_N = 0$ and $0 \leq N \leq 15$;
 (CS) <-- 0001 if $DO_N = 1$ and $N = 0$;
 (CS) <-- 0100 if $DO_N = 1$ and $1 \leq N \leq 15$;

REGISTERS AFFECTED: CS

5.6 Test and set bit.

ADDR	MODF.	MNEMONIC	FORMAT/OPCODE			
			8	4	4	16
D		TSB N, ADDR				
DX		TSB N, ADDR, RX	59	N	RX	ADDR

DESCRIPTION: Bit number N ($0 \leq N \leq 15$) of the Derived Operand, DO, is tested and set to one. The CS is set according to the test.

Note: External memory accesses shall be inhibited until this instruction is complete.

REGISTER TRANSFER DESCRIPTION:

(CS) $\leftarrow$ 0010 and $(DO_N) \leftarrow 1$ if $DO_N = 0$ and $0 \leq N \leq 15$;
 (CS) $\leftarrow$ 0001 if $(DO_N) = 1$ and $N = 0$;
 (CS) $\leftarrow$ 0100 if $(DO_N) = 1$ and $1 \leq N \leq 15$;

REGISTERS AFFECTED: CS

MIL-STD-1750A (USAF)
2 July 1980

5.7 Set variable bit in register.

ADDR MODE MNEMONIC

FORMAT/OPCODE

R SVBR RA, RB

8	4	4
5A	RA	RB

DESCRIPTION: Bit number N ($0 \leq N \leq 15$) of the register RB is set to one where the least significant four bits of the contents of register RA is N. Bits $(RA)_{0-11}$ have no effect on the operation. If $RA = RB$, then the count is determined first and then the appropriate bit is changed.

REGISTER TRANSFER DESCRIPTION:

$(RB)_N \leftarrow 1$ where $N = (RA)_{12-15}$;

REGISTERS AFFECTED: RB

5.8 Reset variable bit in register.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4

R		RVBR RA, RB	5C RA RB

DESCRIPTION: Bit number N ($0 \leq N \leq 15$) of register RB is set to zero where the least significant four bits of the contents of register RA is N . Bits $(RA)_{0-11}$ have no effect on the operation. If $RA = RB$, then the count is determined first and then the appropriate bit is changed.

REGISTER TRANSFER DESCRIPTION:

$(RB)_N \leftarrow 0$ where $N = (RA)_{12-15}$:

REGISTERS AFFECTED: RB

MIL-STD-1750A (USAF)
2 July 1980

5.9 Test variable bit in register.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE		
			8	4	4

R		TVBR RA, RB	5E	RA	RB

DESCRIPTION: Bit number N ($0 \leq N \leq 15$) of register RB is tested where the least significant four bits of the contents of register RA is N . The Condition Status, CS, is then set to indicate non-zero if bit number N of register RB is a one. Otherwise, CS is set to indicate zero.

REGISTER TRANSFER DESCRIPTION:

$N = (RA)_{12-15}$

(CS) $\leftarrow$ 0010 if $(RB_N) = 0$ and $0 \leq N \leq 15$;

(CS) $\leftarrow$ 0001 if $(RB_N) = 1$ and $N = 0$;

(CS) $\leftarrow$ 0100 if $(RB_N) = 1$ and $1 \leq N \leq 15$;

REGISTERS AFFECTED: CS

5.10 Shift left logical.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
R	SLL	RB, N	60 N-1 RB 1 ≤ N ≤ 16

DESCRIPTION: The contents of the Derived Address, DA (i.e., the contents of register RB) are shifted left logically N positions. The shifted result is stored in RB. The logical shift left operation is as follows: zeros enter the least significant bit position (bit 15) and bits shifted out of the sign bit position (bit 0) are lost. The condition status, CS, is set based on the result in register RB.

Note: N-1 = 0 represents a shift of one position.
N-1 = 15 represents a shift of sixteen positions.

	0	15
EXAMPLE: RB Before Shift	sabc defg hijk lmp	
RB After Shift (N=4)	defg hijk lmp 0000	

REGISTER TRANSFER DESCRIPTION:

{RB} <-- {RB} Shifted left logically by N positions;

{CS} <-- 0010 if {RB} = 0;
{CS} <-- 0001 if {RB} < 0;
{CS} <-- 0100 if {RB} > 0;

REGISTERS AFFECTED: RB, CS

MIL-STD-1750A (USAF)
2 July 1980

5.11 Shift right logical.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R	SRL	RB,N	61 N-1 RB 1 ≤ N ≤ 16

DESCRIPTION: The contents of the Derived Address, DA (i.e., the contents of register RB), are shifted right logically N positions. The shifted result is stored in RB. The logical shift right operation is as follows: zeros enter the sign bit position (bit 0) and bits shifted out of the least significant bit position (bit 15) are lost. The condition status, CS, is set based on the result in register RB.

Note: N-1 = 0 represents a shift of one position.

N-1 = 15 represents a shift of sixteen positions.

	0	15
EXAMPLE: RB Before Shift	sabc defg hijk lnnp	
RB After Shift (N=4)	0000 sabc defg hijk	

REGISTER TRANSFER DESCRIPTION:

(RB) <-- (RB) Shifted right logically by N positions;

(CS) <-- 0010 if (RB) = 0;

(CS) <-- 0001 if (RB) < 0;

(CS) <-- 0100 if (RB) > 0;

REGISTERS AFFECTED: RB, CS

5.12 Shift right arithmetic.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
R		SRA RB, N	62 N-1 RB 1 ≤ N ≤ 16

DESCRIPTION: The contents of the Derived Address, DA (i.e., the contents of register RB), are shifted right arithmetically N positions. The shifted result is stored in RB. The arithmetic right shift operation is as follows: the sign bit, which is not changed, is copied into the next position for each position shifted and bits shifted out of the least significant bit position (bit 15) are lost. The condition status, CS, is set based on the result in register RB.

Note: N-1 = 0 represents a shift of one position.

N-1 = 15 represents a shift of sixteen positions.

	0	15
EXAMPLE: RB Before Shift	sabc defg hijk lmp	
RB After Shift (N=4)	ssss sabc defg hijk	

REGISTER TRANSFER DESCRIPTION:

(RB) <-- (RB) Shifted right arithmetically by N positions;

(CS) <-- 0010 if (RB) = 0;

(CS) <-- 0001 if (RB) < 0;

(CS) <-- 0100 if (RB) > 0;

REGISTERS AFFECTED: RB, CS

MIL-STD-1750A (USAF)
2 July 1980

5.13 Shift left cyclic.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE			
			8	4	4	
R		SLC RB,N	63	N-1	RB	1 ≤ N ≤ 16

DESCRIPTION: The contents of the Derived Address, DA (i.e., the contents of register RB), are shifted left cyclically N positions. The shifted result is stored in RB. The cyclic left shift operation is as follows: bits shifted out of the sign bit position (bit 0) enter the least significant bit position (bit 15) and, consequently, no bits are lost. The conditions status, CS, is set based on the result in RB.

Note: N-1 = 0 represents a shift of one position.

N-1 = 15 represents a shift of sixteen positions.

	0	15
EXAMPLE: RB Before Shift	sabc defg hijk lmp	
RB After Shift (N=4)	defg hijk lmp sabc	

REGISTER TRANSFER DESCRIPTION:

(RB) <-- (RB) Shifted left cyclically by N positions;

(CS) <-- 0010 if (RB) = 0;

(CS) <-- 0001 if (RB) < 0;

(CS) <-- 0100 if (RB) > 0;

REGISTERS AFFECTED: RB, CS

5.14 Double shift left logical.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
R		DSLL RB,N	65 N-1 RB 1 ≤ N ≤ 16

DESCRIPTION: The concatenated contents of the Derived Address, DA, and DA+1 (i.e., the concatenated contents of RB and RB+1), are shifted left logically N positions. The shifted results are stored in RB and RB+1. The double left shift logical operation is as follows: zeros enter the least significant bit position of RB+1, bits shifted out of the sign bit position of RB+1 enter the least significant bit of RB and bits shifted out of the sign bit position of RB are lost. The condition status, CS, is set based on the result in registers RB and RB+1.

Note: N-1 = 0 represents a shift of one position.

N-1 = 15 represents a shift of sixteen positions.

EXAMPLE: RB, RB+1 Before Shift

0	RB	15	0	RB+1	15
s ₁ abc	defg	hijk	lmnp	s ₂ qrs	tuvw
xyzz	zzzz				

RB, RB+1 After Shift (N=4)

0	RB	15	0	RB+1	15
defg	hijk	lmnp	s ₂ qrs	tuvw	xyzz
zzzz	0000				

REGISTER TRANSFER DESCRIPTION:

(RB,RB+1) <-- (RB,RB+1) Shifted left logically by N positions;

(CS) <-- 0010 if (RB,RB+1) = 0;

(CS) <-- 0001 if (RB,RB+1) < 0;

(CS) <-- 0100 if (RB,RB+1) > 0;

REGISTERS AFFECTED: RB, RB+1, CS

MIL-STD-1750A (USAF)
2 July 1980

5.15 Double shift right logical.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE		
			8	4	4
R		DSRL RB,N	66	N-1 RB	1 ≤ N ≤ 16

DESCRIPTION: The concatenated contents of the Derived Address, DA, and DA + 1 (i.e., the concatenated contents of RB and RB + 1), are shifted right logically N positions. The shifted results are stored in RB and RB + 1. The double logical right shift operation is as follows: zeros enter the sign bit position of RB, bits shifted out of the least significant bit position of RB enter the sign bit position of RB + 1 and bits shifted out of the least significant bit position of RB + 1 are lost. The condition status, CS, is set based on the result in register RB and RB + 1.

Note: N-1 = 0 represents a shift of one position.

N-1 = 15 represents a shift of sixteen positions.

EXAMPLE: RB, RB+1 Before Shift

0	RB	15	0	RB+1	15
s ₁ abc	defg	hijk	lmnp	s ₂ qrs	tuvw
xyzz	zzzz				

RB, RB+1 After Shift (N=4)

0	RB	15	0	RB+1	15
0000	s ₁ abc	defg	hijk	lmp	s ₂ qrs
tuvw	xyzz				

REGISTER TRANSFER DESCRIPTION:

(RB,RB+1) <-- (RB,RB+1) Shifted right logically by N positions;

(CS) <-- 0010 if (RB,RB+1) = 0;

(CS) <-- 0001 if (RB,RB+1) < 0;

(CS) <-- 0100 if (RB,RB+1) > 0;

REGISTERS AFFECTED: RB, RB+1, CS

5.16 Double shift right arithmetic.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
R		DSRA RB,N	67 N-1 RB 1 ≤ N ≤ 16

DESCRIPTION: The concatenated contents of the Derived Address, DA, and DA + 1 (i.e., the concatenated contents of RB and RB + 1), are shifted right arithmetically N positions. The shifted results are stored in RB and RB + 1. The double right shift arithmetic operation is as follows: the sign bit of RB, which is not changed, is copied into the next position for each position shifted, bits shifted out of the least significant position of RB enter the sign bit position of RB + 1, and bits shifted out of the least significant bit position of RB + 1 are lost. The condition status, CS, is set based on the result in register RB and RB + 1.

Note: N-1 = 0 represents a shift of one position.

N-1 = 15 represents a shift of sixteen positions.

EXAMPLE: RB, RB+1 Before Shift

0	RB	15	0	RB+1	15
s ₁ abc	defg	hijk	lmnp	s ₂ qrs	tuvw xyzz zzzz

RB, RB+1 After Shift (N=4)

0	RB	15	0	RB+1	15
s ₁ s ₁ s ₁ s ₁	s ₁ abc	defg	hijk	lmnp	s ₂ qrs tuvw xyzz

REGISTER TRANSFER DESCRIPTION:

(RB,RB+1) <-- (RB,RB+1) Shifted right arithmetically by N positions;

(CS) <-- 0010 if (RB,RB+1) = 0;

(CS) <-- 0001 if (RB,RB+1) < 0;

(CS) <-- 0100 if (RB,RB+1) > 0;

REGISTERS AFFECTED: RB, RB+1, CS

MIL-STD-1750A (USAF)
2 July 1980

5.17 Double shift left cyclic.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE		
			8	4	4
R		DSLCL RB,N	68	N-1	RB 1 ≤ N ≤ 16

DESCRIPTION: The concatenated contents of the Derived Address, DA, and DA + 1 (i.e., the concatenated contents of RB and RB + 1), are shifted left cyclically N positions. The shifted results are stored in RB and RB + 1. The double left shift cyclic operation is as follows: bits shifted out of the sign bit position of RB enter the least significant bit position of RB + 1, bits shifted out of the sign bit position of RB + 1 enter the least significant bit position of RB, and, consequently, no bits are lost. The condition status, CS, is set based on the result in RB and RB + 1.

Note: N-1 = 0 represents a shift of one position.

N-1 = 15 represents a shift of sixteen positions.

EXAMPLE: RB, RB+1 Before Shift

0	RB	15	0	RB+1	15
s ₁ abc	defg	hijk	lmnp	s ₂ qrs	tuvw xyz zzzz

RB, RB+1 After Shift (N=4)

0	RB	15	0	RB+1	15
defg	hijk	lmnp	s ₂ qrs		tuvw xyz zzzz s ₁ abc

REGISTER TRANSFER DESCRIPTION:

(RB,RB+1) <-- (RB,RB+1) Shifted left cyclically by N positions;

(CS) <-- 0010 if (RB,RB+1) = 0;

(CS) <-- 0001 if (RB,RB+1) < 0;

(CS) <-- 0100 if (RB,RB+1) > 0;

REGISTERS AFFECTED: RB, RB+1, CS

DSLCL

AD-A100 577

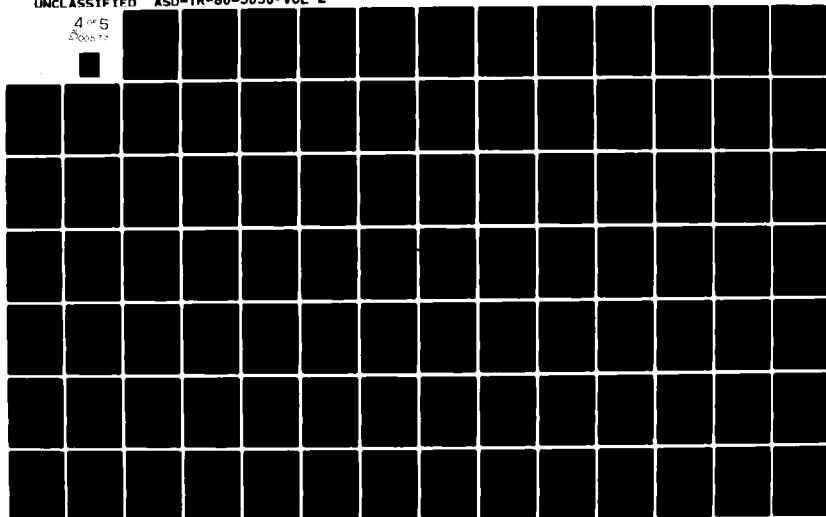
AERONAUTICAL SYSTEMS DIV WRIGHT-PATTERSON AFB OH
AFSC STANDARDIZATION CONFERENCE, 1553, 1589, 1750, 1760, ADA, N--ETC(U)
NOV 80 E C GANGH, S E SMITH
ASD-TR-80-5050-VOL-2

F/G 1/3

NL

UNCLASSIFIED

4 of 5
2005 12



5.18 Shift logical count in register.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R		SLR RA, RB	6A RA RB $(RB) \leq 16$

DESCRIPTION: The contents of register RA are shifted logically N positions, where N is the contents of register RB. If N is positive ($(RB)_0 = 0$), then the shift direction is left; if N is negative (2's complement notation, $(RB)_0 = 1$), then the shift direction is right. The condition status, CS, is set based on the result in RA.

Note: N = 0 represents a shift of zero positions.

If $|N| > 16$, the fixed point overflow occurs, no shifting takes place, and this instruction is treated as a NOP (see page 137).

The contents of RB remain unchanged, unless RA = RB; in this event the contents are shifted N positions.

(See "Description" of the logical shift instructions, SLL and SRI. (see pages 41 and 42), for the definition of shift operations.)

REGISTER TRANSFER DESCRIPTION:

$PI_4 \leftarrow 1$, exit, if $|N| > 16$;

$(RA) \leftarrow (RA)$ Shifted left logically by (RB) positions,

if $0 < (RB) \leq 16$;

$(RA) \leftarrow (RA)$ Shifted right logically by $-(RB)$ positions,

if $0 > (RB) \geq -16$;

$(CS) \leftarrow 0010$ if $(RA) = 0$;

$(CS) \leftarrow 0001$ if $(RA) < 0$;

$(CS) \leftarrow 0100$ if $(RA) > 0$;

REGISTERS AFFECTED: RA, RB, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.19 Shift arithmetic count in register.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE			
			8	4	4	
R		SAR RA, RB	6B	RA	RB	(RB) ≤ 16

DESCRIPTION: The contents of register RA are shifted arithmetically N positions, where N is the contents of register RB. If N is positive ((RB₀) = 0), then the shift direction is left; if N is negative (2's complement notation, (RB₀) = 1), then the shift direction is right. The condition status, CS, is set based on the result in RA.

Note: N = 0 represents a shift of zero positions.

If |N| > 16, the fixed point overflow occurs, no shifting takes place, and this instruction is treated as a NOP (see page 137).

The contents of RB remain unchanged, unless RA = RB; in this event, the contents are shifted N positions.

(See "Description" of the arithmetic shift instruction SRA (see page 43) for definition of the right shift operation. Left shift causes "zeros" to be shifted into low order position of result.)

Fixed point overflow occurs if the sign bit changes during a left shift.

REGISTER TRANSFER DESCRIPTION:

PI₄ ← 1, exit, if |N| > 16;

(RA) ← (RA) Shifted left arithmetically (RB) positions,

if 16 ≥ (RB) > 0;

(RA) ← (RA) Shifted right arithmetically -(RB) positions,

if 0 > (RB) ≥ -16;

PI₄ ← 1, if (RA₀) changes during the shift;

(CS) ← 0010 if (RA) = 0;

(CS) ← 0001 if (RA) < 0;

(CS) ← 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, RB, CS, PI

5.20 Shift cyclic, count in register.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R	SCR	RA, RB	6C RA RB (RB) ≤ 16

DESCRIPTION: The contents of register RA are shifted cyclically N positions, where N is the contents of register RB. If N is positive ((RB)₀ = 0), then the shift direction is left; if N is negative (2's complement notation, (RB)₀ = 1), then the shift direction is right. The condition status, CS, is set based on the result in RA.

Note: N = 0 represents a shift of zero positions.

If |N| > 16, the fixed point overflow occurs, no shifting takes place, and this instruction is treated as a NOP (see page 137).

(See "Description" of the cyclic shift instruction, SLC (see page 44), for definition of shift operations.)

The contents of RB remain unchanged, unless RA = RB in this event, the contents are shifted N positions.

REGISTER TRANSFER DESCRIPTION:

PI₄ <-- 1, exit, if |N| > 16;

(RA) <-- (RA) Shifted left cyclically by (RB) positions.

if 0 < (RB) ≤ 16;

(RA) <-- (RA) Shifted right cyclically by -(RB) positions.

if 0 > (RB) ≥ -16;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, RB, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.21 Double shift logical count in register.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE			
			8	4	4	
R		DSL R RA, RB	6D	RA	RB	(RB) ≤ 32

DESCRIPTION: The concatenated contents of registers RA and RA + 1 are shifted logically N positions where register RB contains the count, N. If the count is positive ((RB₀) = 0), then the shift direction is left. If the count is negative (2's complement notation, (RB₀) = 1), then the shift direction is right. The condition status, CS, is set based on the result in RA and RA + 1.

Note: N = 0 represents a shift of zero positions.

If |N| > 32, the fixed point overflow occurs, no shifting occurs, and this instruction is treated as a NOP (see page 137).

(See "Description" of the double shift logical instructions, DSR1 and DSI1, (see pages 46 and 45), for definition of shift operations.)

The contents of RB remain unchanged, unless RA = RB; in this event, the contents are shifted N positions.

REGISTER TRANSFER DESCRIPTION:

PI₄ ← 1, exit, if |N| > 32;

(RA, RA+1) ← (RA, RA+1) Shifted left logically by (RB) positions

if 32 ≥ (RB) > 0;

(RA, RA+1) ← (RA, RA+1) Shifted right logically by -(RB) positions

if 0 > (RB) ≥ -32;

(CS) ← 0010 if (RA, RA+1) = 0;

(CS) ← 0001 if (RA, RA+1) < 0;

(CS) ← 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, RB, CS, PI

5.22 Double shift arithmetic count in register.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4

R		DSAR RA, RB	6E RA RB (RB) ≤ 32

DESCRIPTION: The concatenated contents of register RA and RA + 1 are shifted arithmetically N positions where register RB contains the count, N. If the count is positive ((RB₀) = 0), then the shift direction is left. If the count is negative (2's complement notation, (RB₀) = 1), then the shift direction is right. The condition status, CS, is set based on the result in RA and RA + 1.

Note: N = 0 represents a shift of zero positions.

If |N| > 32, the fixed point overflow occurs, no shifting occurs, and this instruction is treated as a NOP (see page 137).

The contents of RB remain unchanged, unless RA = RB; in this event, the contents are shifted N positions.

(See "Description" of the double shift arithmetic instruction, DSRA (see page 47), for the definition of the right shift operation. Left shift causes "zeros" to be shifted into low order position of result.)

Fixed point overflow occurs if the sign bit is changed during a left shift.

REGISTER TRANSFER DESCRIPTION:

PI₄ ← 1, exit, if |N| > 32;

(RA, RA+1) ← (RA, RA+1) Shifted left arithmetically (RB) positions,

if 32 ≥ (RB) > 0;

(RA, RA+1) ← (RA, RA+1) Shifted right arithmetically -(RB) positions,

if 0 > (RB) ≥ -32;

PI₄ ← 1, if (RA₀) changes during the shift;

(CS) ← 0010 if (RA, RA+1) = 0;

(CS) ← 0001 if (RA, RA+1) < 0;

(CS) ← 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, RB, CS, PI

MIL-STD-1750A (USAF)

2 July 1980

5.23 Double shift cyclic count in register.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R		DSCR RA, RB	6F RA RB (RB) ≤ 32

DESCRIPTION: The concatenated contents of registers RA and RA + 1 are shifted cyclically N positions, where register RB contains the count, N. If the count is positive ((RB₀) = 0), the shift direction is left. If the count is negative (2's complement notation, (RB₀) = 1), the shift direction is right. The condition status, CS, is set based on the result in RA and RA + 1.

Note: N = 0 represents a shift of zero positions.

If |N| > 32, the fixed point overflow occurs, no shifting occurs, and this instruction is treated as a NOP (see page 137).

(See "Description" of the double shift cyclic instruction, DSLC (see page 48), for the definition of shift operations.)

The contents of RB remain unchanged, unless RA = RB; in this event, the contents are shifted N positions.

REGISTER TRANSFER DESCRIPTION:

PI₄ <-- 1, exit, if |N| > 32;

(RA, RA+1) <-- (RA, RA+1) Shifted left cyclically by (RB) positions

if 32 ≥ (RB) > 0;

(RA, RA+1) <-- (RA, RA+1) Shifted right cyclically by -(RB) positions

if 0 > (RB) ≥ -32;

(CS) <-- 0010 if (RA, RA+1) = 0;

(CS) <-- 0001 if (RA, RA+1) < 0;

(CS) <-- 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, RB, CS, PI

5.24 Jump on condition.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE			
			8	4	4	16
D	JC	C, LABEL	-----			
DX	JC	C, LABEL, RX	70	C	RX	LABEL

			8	4	4	16
I	JCI	C, ADDR	-----			
IX	JCI	C, ADDR, RX	71	C	RX	ADDR

DESCRIPTION: This is a conditional jump instruction wherein the instruction sequence jumps to the Derived Address, DA, if a logical one results from the following operation:

- (1) The 4-bit C field is bit-by-bit ANDed with the 4-bit condition status, CS
- (2) The resulting 4-bits are ORed together
- (3) or if C = 7 or C = F:

Otherwise, the next sequential instruction is executed.

Condition Code

C ₂	C ₁₆	Jump Condition	Mnemonic		
0000	0	NOP	-	-	-
0001	1	less than (zero)	LT	LZ	M
0010	2	equal to (zero)	EQ	EZ	-
0011	3	less than or equal to (zero)	LE	LEZ	NP
0100	4	greater than (zero)	GT	GZ	P
0101	5	not equal to (zero)	NE	NZ	-
0110	6	greater than or equal to (zero)	GE	GEZ	NM
0111	7	unconditional	-	-	-
1000	8	carry	CY	-	-
1001	9	carry or LT	-	-	-
1010	A	carry or EQ	-	-	-
1011	B	carry or LE	-	-	-
1100	C	carry or GT	-	-	-
1101	D	carry or NE	-	-	-
1110	E	carry or GE	-	-	-
1111	F	unconditional	-	-	-

MIL-STD-1750A (USAF)
2 July 1980

REGISTER TRANSFER DESCRIPTION:

(IC) <-- DA if C = 7, or

if C = F, or

if $(C_0 \uparrow CS_0) \vee (C_1 \uparrow CS_1) \vee (C_2 \uparrow CS_2) \vee (C_3 \uparrow CS_3) = 1$;

REGISTERS AFFECTED: IC (if jump is executed)

5.25 Jump to subroutine.

<u>ADDR</u>	<u>MODE</u>	<u>MINI MONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	16
D	JS	RA, LABEL	-----			
DX	JS	RA, LABEL, RX	72	RA	RX	LABEL

DESCRIPTION: The value of the instruction counter (the address of the next sequential instruction) is stored into register RA. Then, the IC is set to the derived address, DA, thus effecting the jump. This sets up the return from subroutine to the address stored in the register RA, i.e., an indexed unconditional jump from location zero using RA as the index register shall transfer control to the instruction following the JS instruction.

Note: If RA == RX, then the derived address, DA, is calculated before the IC is stored in RA.

REGISTER TRANSFER DESCRIPTION:

(RA) <-- (IC);

(IC) <-- DA;

REGISTERS AFFECTED: RA, IC

MTL-STD-1750A (USAF)
2 July 1980

5.26 Subtract one and jump.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	16
D	SOJ	RA, LABEL	-----			
DX	SOJ	RA, LABEL, RX	73	RA	RX	LABEL

DESCRIPTION: The 16 bit contents of register RA are decremented by one. Then if the content of register RA is zero, the next sequential instruction is executed. If the content of register RA is non-zero, then a jump to the Derived Address, DA, occurs.

Note: If RA = RX, then the derived address, DA, is calculated before RA is decremented.

REGISTER TRANSFER DESCRIPTION:

(RA) <-- (RA) - 1;

(IC) <-- DA if (RA) ≠ 0;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS, IC (if the jump is executed)

5.27 Branch unconditionally.

ADDR MODE MNEMONIC

FORMAT/OPCODE

ICR BR LABEL

8		8	
74	D	-128 ≤ D ≤ 127	

DESCRIPTION: A program branch is made to LABEL, i.e., the Derived Address, DA.

REGISTER TRANSFER DESCRIPTION:

(IC) <-- DA ;

REGISTERS AFFECTED: IC

MIL-STD-1750A (USAF)
2 July 1980

5.28 Branch if equal to (zero).

<u>ADDR</u> <u>MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>		
			8	8	
ICR	BEZ	LABEL	75	D	-128 ≤ D ≤ 127

DESCRIPTION: A program branch is made to LABEL, i.e., the Derived Address, DA, if the condition status, CS, indicates that the previous result which set the CS is equal to (zero). Otherwise, the next sequential instruction is executed.

REGISTER TRANSFER DESCRIPTION:

(IC) <-- DA if (CS) = X010;

REGISTERS AFFECTED: IC (if the jump is executed)

5.29 Branch if less than (zero).

ADDR MODE MNEMONIC

FORMAT/OPCODE

			8	8	
ICR	BLT	LABEL	76	D	-128 ≤ D ≤ 127

DESCRIPTION: A program branch is made to LABEL, i.e., the Derived Address, DA, if the condition status, CS, indicates that the previous result which set the CS is less than (zero). Otherwise, the next sequential instruction is executed.

REGISTER TRANSFER DESCRIPTION:

(IC) <-- DA if (CS) = X001;

REGISTERS AFFECTED: IC (if the jump is executed)

MIL-STD-1750A (USAF)
2 July 1980

5.30 Branch to executive.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>		
			8	4	4
S		BEX N	77	0000 N	

DESCRIPTION: This instruction provides a means to jump to a routine in another address state, AS. It is typically used to make controlled, protected calls to an executive. The 4-bit literal N selects one of 16 executive entry points to be used. Execution of this instruction causes an interrupt to occur using the EXEC call interrupt vector (interrupt 5). The new IC is loaded from the Nth location following the SW in the new processor state. The linkage pointer (LP), service pointer (SVP), and the new processor state (new MK, new SW, and new IC) are fetched from address state zero. The current processor state (old MK, old SW, and old IC) are stored in the address state specified by the new SW AS field. Interrupts are disabled when BEX is executed. The EXEC call interrupt cannot be masked or disabled. Arguments associated with the BEX instruction are passed by software convention. The processor lock and key function is ignored when this instruction is executed. An attempt to branch into an execute protected area of memory shall result in PT_0 being set to 1.

REGISTER TRANSFER DESCRIPTION:

$(RQ, RQ+1, RQ+2) \leftarrow (MK, SW, IC);$

$(SVP) \leftarrow [2B_{16}], \text{ where } AS = 0;$

$PI_5 \leftarrow 1;$

$(MK, SW, IC) \leftarrow [(SVP), (SVP)+1, (SVP)+2+N], \text{ where } AS = 0;$

$(LP) \leftarrow [2A_{16}], \text{ where } AS = 0;$

$[(LP), (LP)+1, (LP)+2] \leftarrow (RQ, RQ+1, RQ+2), \text{ where } AS = SW_{12-15};$

REGISTERS AFFECTED: MK, SW, IC, PI

5.31 Branch if less than or equal to (zero).

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>		
			8	8	

ICR	BLE	LABEL	78	0	-128 ≤ D ≤ 127

DESCRIPTION: A program branch is made to LABEL, i.e., the Derived Address, DA, if the condition status, CS, indicates that the previous result which set the CS is less than or equal to (zero). Otherwise, the next sequential instruction is executed.

REGISTER TRANSFER DESCRIPTION:

(IC) <-- DA if (CS) = X010 or (CS) = X001;

REGISTERS AFFECTED: IC (if the jump is executed)

MIL-STD-1750A (USAF)
2 July 1980

5.32 Branch if greater than (zero).

<u>ADDR</u> <u>MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>	
			8	8
ICR	BGT	LABEL	79	D -128 ≤ D ≤ 127

DESCRIPTION: A program branch is made to LABEL, i.e., the Derived Address, DA, if the condition status, CS, indicates that the previous result which set the CS is greater than (zero). Otherwise, the next sequential instruction is executed.

REGISTER TRANSFER DESCRIPTION:

(IC) <-- DA if (CS) = X100;

REGISTERS AFFECTED: IC (if the jump is executed)

5.33 Branch if not equal to (zero).

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>		
			8	8	
ICR	BNZ	LABEL	7A	D	-128 ≤ D ≤ 127

DESCRIPTION: A program branch is made to LABEL, i.e., the Derived Address, DA, if the condition status, CS, indicates that the previous result which set the CS is not equal to (zero). Otherwise, the next sequential instruction is executed.

REGISTER TRANSFER DESCRIPTION:

(IC) ← DA if (CS) = X100 or (CS) = X001;

REGISTERS AFFECTED: IC (if the jump is executed)

MIL-STD-1750A (USAF)
2 July 1980

5.34 Branch if greater than or equal to (zero).

<u>ADDR</u> <u>MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>	
			8	8

ICR	BGE	LABEL	7B	D -128 ≤ D ≤ 127

DESCRIPTION: A program branch is made to LABEL, i.e., the Derived Address, DA, if the condition status, CS, indicates that the previous result which set the CS is greater than or equal to (zero). Otherwise, the next sequential instruction is executed.

REGISTER TRANSFER DESCRIPTION:

(IC) <-- DA if (CS) = X100 or (CS) = X010;

REGISTERS AFFECTED: IC (if the jump is executed)

5.35 Load status.

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	16
D	LST	ADDR	-----			
DX	LST	ADDR, RX	7D	0000	RX	ADDR

			8	4	4	16
I	LSTI	ADDR	-----			
IX	LSTI	ADDR, RX	7C	0000	RX	ADDR

DESCRIPTION: 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. This is a privileged instruction.

Note: This instruction is an unconditional jump and is typically used to exit from an interrupt routine. DA, DA + 1, and DA + 2, in this typical case, contain the Interrupt Mask, Status Word, and Instruction Counter values for the interrupted program and the execution of LST causes the program to return to its status prior to being interrupted.

REGISTER TRANSFER DESCRIPTION:

(MK, SW, IC) <-- [DA, DA+1, DA+2];

REGISTERS AFFECTED: MK, SW, IC

MIL-STD-1750A (USAF)
2 July 1980

5.36 Stack IC and jump to subroutine.

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	16
D	SJS	RA, LABEL	-----			
DX	SJS	RA, LABEL, RX	7E	RA	RX	LABEL

DESCRIPTION: The contents of register RA are decremented by one. 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 Derived Address, DA. RA is the stack pointer and can be selected by the programmer as any one of the 16 general registers.

Note: If $RA = RX$, then the derived address, DA, is calculated before RA is decremented.

REGISTER TRANSFER DESCRIPTION:

$(RA) \leftarrow (RA) - 1;$

$[(RA)] \leftarrow (IC);$

$(IC) \leftarrow DA;$

REGISTERS AFFECTED: IC, RA

5.37 Unstack IC and return from subroutine.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	
S		URS RA	7F	RA	0	

DESCRIPTION: The contents of the memory location pointed to by register RA is loaded into the instruction counter, IC. RA is then incremented by one. Any one of the 16 general registers may be designated as the stack pointer. This instruction is the subroutine return for SJS, Stack and Jump to Subroutine.

REGISTER TRANSFER DESCRIPTION:

(IC) <-- [(RA)];

(RA) <-- (RA) + 1;

REGISTERS AFFECTED: RA, IC

MIL-STD-1750A (USAF)
2 July 1980

5.38 Single precision load.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
R	LR	RA, RB	844 81 RA RB
B	LB	BR, DSPL	4228 0 0 BR' DSPL $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
BX	LBX	BR, RX	42244 4 0 BR' 0 RX $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
ISP	LISP	RA, N	844 82 RA N-1 $1 \leq N \leq 16$
ISN	LISN	RA, N	844 83 RA N-1 $1 \leq N \leq 16$
D DX	L L	RA, ADDR RA, ADDR, RX	84416 80 RA RX ADDR
IM IMX	LIM LIM	RA, DATA RA, DATA, RX	84416 85 RA RX DATA
I IX	LI LI	RA, ADDR RA, ADDR, RX	84416 84 RA RX ADDR

DESCRIPTION: The single precision Derived Operand, DO, is loaded into the register RA. The Condition Status, CS, is set based on the result in register RA.

MIL-STD-1750A (USAF)
2 July 1980

REGISTER TRANSFER DESCRIPTION:

(RA) <-- DO;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS

MIL-STD-1750A (USAF)
2 July 1980

5.39 Double precision load.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
R	DLR	RA, RB	844 87 RA RB
B	DLB	BR, DSPL	4228 0 1 BR' DSPL $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
BX	DLBX	BR, RX	42244 4 0 BR' 1 RX $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
D DX	DL DL	RA, ADDR RA, ADDR, RX	84416 86 RA RX ADDR
I IX	DLI DLI	RA, ADDR RA, ADDR, RX	84416 88 RA RX ADDR

DESCRIPTION: The double precision Derived Operand, DO, is loaded into the register RA and RA + 1 such that the MSH of DO is in RA. The Condition Status, CS, is set based on the result in RA and RA + 1.

REGISTER TRANSFER DESCRIPTION:

(RA, RA+1) <-- DO;

(CS) <-- 0010 if (RA, RA+1) = 0 (Double fixed point zero);
 (CS) <-- 0001 if (RA, RA+1) < 0;
 (CS) <-- 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, CS

5.40 Load multiple registers.

ADDR MODE		MMNEMONIC	FORMAT/OPCODE			
			8	4	4	16
D	LM	N, ADDR				
DX	LM	N, ADDR, RX	89	N	RX	ADDR
0 ≤ N ≤ 15						

DESCRIPTION. The contents of the Derived Address, DA, are loaded into register R0, then the contents of the DA + 1 are loaded into register R1, ..., finally, the contents of DA + N are loaded into RN. Effectively this instruction allows the transfer of (N + 1) words from memory to the register file.

REGISTER TRANSFER DESCRIPTION:

(R0) <-- [DA] ;

(R1) <-- [DA+1];

(R2) <-- [DA+2];

.

(RN) <-- [DA+N];

REGISTERS AFFECTED: R0 through RN

MIL-STD-1750A (USAF)
2 July 1980

5.41 Extended precision floating point load.

<u>ADDR</u> <u>MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	16
D	EFL	RA, ADDR	-----			
DX	EFL	RA, ADDR, RX	8A	RA	RX	ADDR

DESCRIPTION: The extended precision floating point Derived Operand, DO, is loaded into registers RA, RA + 1, and RA + 2 such that the most significant 16-bits of the word are loaded into register RA. The condition status, CS, is set based on the results in registers RA, RA + 1, and RA + 2.

REGISTER TRANSFER DESCRIPTION:

(RA, RA+1, RA+2) <-- DO;

(CS) <-- 0010 if (RA, RA+1, RA+2) = 0;

(CS) <-- 0001 if (RA, RA+1, RA+2) < 0;

(CS) <-- 0100 if (RA, RA+1, RA+2) > 0;

REGISTERS AFFECTED: RA, RA+1, RA+2, CS

5.42 Load from upper byte.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE			
			8	4	4	16
D	LUB	RA, ADDR	-----			
DX	LUB	RA, ADDR, RX	8B	RA	RX	ADDR

			8	4	4	16
I	LUBI	RA, ADDR	-----			
IX	LUBI	RA, ADDR, RX	8D	RA	RX	ADDR

DESCRIPTION: The MSH (upper byte) of the Derived Operand, DO, is loaded into the LSH (lower byte) of register RA. The MSH (upper byte) of RA is unaffected. The condition status, CS, is set based on the result in RA.

REGISTER TRANSFER DESCRIPTION:

(RA)₈₋₁₅ <-- DO₀₋₇;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS

MIL-STD-1750A (USAF)
2 July 1980

5.43 Load from lower byte.

<u>ADDR MODE</u> <u>MNEMONIC</u>			<u>FORMAT/OPCODE</u>			
			8	4	4	16
D	LLB	RA, ADDR	-----			
DX	LLB	RA, ADDR, RX	8C	RA	RX	ADDR

			8	4	4	16
I	LLBI	RA, ADDR	-----			
IX	LLBI	RA, ADDR, RX	8E	RA	RX	ADDR

DESCRIPTION: The LSH (lower byte) of the Derived Operand, DO, is loaded into the LSH (lower byte) of register RA. The MSH (upper byte) of RA is unaffected. The condition status, CS, is set based on the result in RA.

REGISTER TRANSFER DESCRIPTION:

(RA)₈₋₁₅ <-- DO₈₋₁₅;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS

5.44 Pop multiple registers off the stack.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
S		POPM RA, RB	8F RA RB

DESCRIPTION: For $RA \leq RB$, registers RA through RB are loaded sequentially from a stack in memory using R15 as the stack pointer.

For $RA > RB$, registers RA through R14 and then R0 through RB are loaded sequentially from the stack.

In both cases,

- as each word is popped from the stack, R15 is incremented by one;
- if R15 is included in the transfer, then it is effectively ignored;
- on completion, R15 points to the top word of the stack remaining.

REGISTER TRANSFER DESCRIPTION:

```

if RA ≤ RB then
  for i = 0 thru RB - RA do
    begin
      if RA + i ≠ 15 then (RA + i) ← [(R15)];
      (R15) ← (R15) + 1;
    end;
else
  begin
    for i = 0 thru 15 - RA do
      begin
        if RA + i ≠ 15 then (RA + i) ← [(R15)];
        (R15) ← (R15) + 1;
      end;
    for i = 0 thru RB do
      begin
        (i) ← [(R15)];
        (R15) ← (R15) + 1;
      end;
  end;

```

REGISTERS AFFECTED: RA through R14, R0 through RB, R15

MIL-STD-1750A (USAF)
2 July 1980

5.45 Single precision store.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE			
			4	2	2	8
B	STB	BR, DSPL	-----			
			0 2 BR' DSPL			12 ≤ BR ≤ 16 BR' = BR - 12 RA = R2

			4	2	2	4 4
BX	STBX	BR, RX	-----			
			4 0 BR' 2 RX			12 ≤ BR ≤ 16 BR' = BR - 12 RA = R2

			8	4	4	16
D	ST	RA, ADDR	-----			
DX	ST	RA, ADDR, RX	90	RA	RX	ADDR

			8	4	4	16
I	STI	RA, ADDR	-----			
IX	STI	RA, ADDR, RX	94	RA	RX	ADDR

DESCRIPTION: The contents of the register RA are stored into the Derived Address, DA.

REGISTER TRANSFER DESCRIPTION:

[DA] <-- (RA);

REGISTERS AFFECTED: None

5.46 Store a non-negative constant.

<u>ADDR</u> <u>MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
D DX		STC N,ADDR	8	4	4	16
		STC N,ADDR,RX	91	N	RX	ADDR
I IX		STCI N,ADDR	8	4	4	16
		STCI N,ADDR,RX	92	N	RX	ADDR

DESCRIPTION: The constant N, where N is an integer ($0 \leq N \leq 15$) is stored at the Derived Address, DA. For the special case of storing zero into memory the mnemonics

STZ ADDR,RX for direct addressing
and STZI ADDR,RX for indirect addressing

may be used. In this special case, the N field equals 0.

REGISTER TRANSFER DESCRIPTION:

[DA] $\leftarrow$ N, where $0 \leq N \leq 15$;

REGISTERS AFFECTED: None

MIL-STD-1750A (USAF)
2 July 1980

5.47 Move multiple words, memory-to-memory.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4

S		MOV RA, RB	93 RA RB

DESCRIPTION: This instruction allows the memory-to-memory transfer of N words where N is an integer between zero and $2^{16} - 1$ and is represented by the contents of RA + 1. The contents of RB are the address of the first word to be transferred and the contents of RA are the address of where the first word is to be transferred. After each word transfer, RA and RB are incremented, and RA + 1 is decremented.

Note: Any pending interrupts are honored after each single word transfer is completed. The IC points to the current instruction location until the last transfer is completed.

RA has a final value of the last stored address plus one; RA + 1 has a final value of zero.

RB has a final value equal to the address of the last word transferred plus one.

REGISTER TRANSFER DESCRIPTION:

Step 1: [(RA)] <-- [(RB)] if (RA+1) > 0; Go to Step 4 otherwise;

Step 2: (RA) <-- (RA)+1, (RB) <-- (RB)+1, (RA+1) <-- (RA+1)-1;

Step 3: REPEAT STEPS 1 and 2;

Step 4: Set IC to next instruction address;

REGISTERS AFFECTED: RA, RA+1, RB

MOV

S0

5.48 Double precision store.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE	
			4 2 2 8	
B	DSTB	BR, DSPL	0 3 BR' DSPL	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
			4 2 2 4 4	
BX	DSTX	BR, RX	4 0 BR' 3 RX	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
			8 4 4 16	
D	DST	RA, ADDR	96 RA RX ADDR	
DX	DST	RA, ADDR, RX	96 RA RX ADDR	
			8 4 4 16	
I	DSTI	RA, ADDR	98 RA RX ADDR	
IX	DSTI	RA, ADDR, RX	98 RA RX ADDR	

DESCRIPTION: The contents of registers RA and RA + 1 are stored at the Derived Address, DA, and DA + 1, respectively

REGISTER TRANSFER DESCRIPTION:

[DA, DA+1] <-- (RA, RA+1);

REGISTERS AFFECTED: None

MIL-STD-1750A (USAF)
2 July 1980

5.49 Store register through mask.

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>				
			8	4	4	16	
D	SRM	RA, ADDR	-----				
DX	SRM	RA, ADDR, RX	97	RA	RX	ADDR	

DESCRIPTION: The contents of register RA are stored into the Derived Address, DA, through the mask in register RA + 1. For each position in the mask that is a one, the corresponding bit of register RA is stored into the corresponding bit of the DA. For each position in the mask that is a zero no change is made to the corresponding bit stored in the DA.

REGISTER TRANSFER DESCRIPTION:

$[DA] \leftarrow \{[DA] \uparrow (\overline{RA+1})\} \vee \{[RA] \uparrow [RA+1]\};$

$(RA+1) = \text{MASK}, (RA) = \text{DATA};$

or, equivalently,

$(RQ) \leftarrow [DA];$

$(RQ)_i \leftarrow (RA)_i \text{ if } (RA+1)_i = 1 \text{ for } i = 0, 1, \dots, 15;$

$[DA] \leftarrow (RQ);$

REGISTERS AFFECTED: None

5.50 Store multiple registers.

<u>ADDR</u> <u>MODE</u>		<u>MNEMONIC</u>	<u>FORMAT / OPCODE</u>			
			8	4	4	16
D		STM	N, ADDR			
DX		STM	N, ADDR, RX			
			99	N	RX	ADDR

DESCRIPTION: The contents of register R0 are stored into the Derived Address, DA; then the contents of R1 are stored into DA + 1;; finally, the contents of RN are stored into DA + N where N is an integer, $0 \leq N \leq 15$. Effectively, this instruction allows the transfer of (N + 1) words from the register file to memory.

REGISTER TRANSFER DESCRIPTION:

[DA] <-- (R0);

[DA+1] <-- (R1);

[DA+2] <-- (R2);

⋮

[DA+N] <-- (RN) $0 \leq N \leq 15$;

REGISTERS AFFECTED: None

MIL-STD-1750A (USAF)
2 July 1980

5.51 Extended precision floating point store.

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>				
			8	4	4		16
D		EFST RA, ADDR	-----				
DX		EFST RA, ADDR, RX	9A	RA	RX		ADDR

DESCRIPTION: The contents of registers RA, RA + 1, RA + 2 are stored at the Derived Address, DA DA + 1, and DA + 2.

REGISTER TRANSFER DESCRIPTION:

[DA, DA+1, DA+2] <-- (RA, RA+1, RA+2);

REGISTERS AFFECTED: None

5.52 Store into upper byte.

ADDR MODE		MNEMONIC	FORMAT/OPCODE			
D DX		STUB RA, ADDR	8	4	4	16
		STUB RA, ADDR, RX	9B	RA	RX	ADDR
I IX		SUBI RA, ADDR	8	4	4	16
		SUBI RA, ADDR, RX	9D	RA	RX	ADDR

DESCRIPTION: The LSH (lower byte) of register RA is stored into the MSH (upper byte) of the Derived Address, DA. The LSH (lower byte) of the DA is unchanged.

REGISTER TRANSFER DESCRIPTION:

[DA]₀₋₇ ← (RA)₈₋₁₅;

REGISTERS AFFECTED: None

MIL-STD-1750A (USAF)
2 July 1980

5.53 Store into lower byte.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4 16
D		STLB RA, ADDR	-----
DX		STLB RA, ADDR, RX	9C RA RX ADDR

			8 4 4 16
I		SLBI RA, ADDR	-----
IX		SLBI RA, ADDR, RX	9E RA RX ADDR

DESCRIPTION: The LSH (lower byte) of register RA is stored into the LSH (lower byte) of the Derived Address, DA. The MSH (upper byte) of the DA is unchanged.

REGISTER TRANSFER DESCRIPTION:

$[DA]_{8-15} \leftarrow (RA)_{8-15}$:

REGISTERS AFFECTED: None

5.54 Push multiple registers onto the stack.

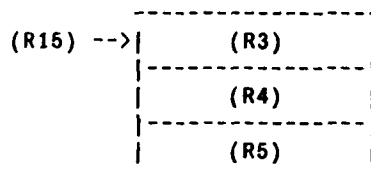
ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
S		PSHM RA, RB	9F RA RB

DESCRIPTION: For $RA \leq RB$, the contents of RB through RA are pushed onto a stack in memory using R15 as the stack pointer. As each register contents are pushed onto the memory stack, R15 is decremented by one word for each word pushed. On completion, R15 points to the last item on the stack, the contents of RA.

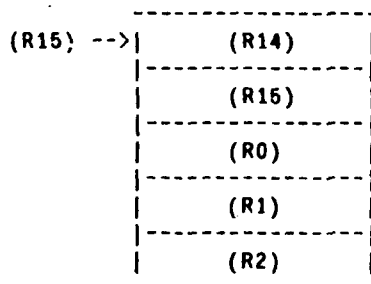
For $RA > RB$, the contents of RB through R0, and then the contents of R15 through RA, are pushed onto the stack. On completion, R15 points to the last item on the stack, the contents of RA.

In both cases, successive increasing addresses on the stack correspond to successive increasing register addresses, with a point discontinuity between R15 and R0 in the latter case.

EXAMPLE: PSHM R3,R5 results in



PSHM R14,R2 results in



MIL-STD-1750A (USAF)
2 July 1980

REGISTER TRANSFER DESCRIPTION:

```
if RA ≤ RB then
    for i = 0 thru RB - RA do
        begin
            (R15) <-- (R15) - 1;
            [(R15)] <-- (RB - i);
        end;
else
    begin
        for i = 0 thru RB do
            begin
                (R15) <-- (R15) - 1;
                [(R15)] <-- (RB - i);
            end;
        for i = 0 thru 15 - RA do
            begin
                (R15) <-- (R15) - 1;
                [(R15)] <-- (R15 - i);
            end;
        end;
end;
```

REGISTERS AFFECTED: R15

5.55 Single precision integer add.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE	
R	AR	RA, RB	8 4 4 A1 RA RB	
B	AB	BR, DSPL	4 2 2 8 1 0 BR' DSPL	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
BX	ABX	BR, RX	4 2 2 4 4 4 0 BR' 4 RX	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
ISP	AISP	RA, N	8 4 4 A2 RA N-1	$1 \leq N \leq 16$
D DX	A A	RA, ADDR RA, ADDR, RX	8 4 4 16 A0 RA RX ADDR	
IM	AIM	RA, DATA	8 4 4 16 4A RA 1 DATA	

DESCRIPTION: The Derived Operand (DO) is added to the contents of the RA register. The result (a 2's complement sum) is stored in register RA. The condition status (CS) is set based on the result in register RA and carry. A fixed point overflow occurs if both operands are of the same sign and the sum is of opposite sign.

MIL-STD-1750A (USAF)
2 July 1980

REGISTER TRANSFER DESCRIPTION:

$(RA)^2 \leftarrow (RA)^1 + DO;$

$PI_4 \leftarrow 1, \text{ if } (RA_0)^1 = DO_0 \text{ and } (RA_0)^1 \neq (RA_0)^2$

(CS) $\leftarrow$ 0010 if carry = 0 and (RA) = 0;

(CS) $\leftarrow$ 0001 if carry = 0 and (RA) < 0;

(CS) $\leftarrow$ 0100 if carry = 0 and (RA) > 0;

(CS) $\leftarrow$ 1010 if carry = 1 and (RA) = 0;

(CS) $\leftarrow$ 1001 if carry = 1 and (RA) < 0;

(CS) $\leftarrow$ 1100 if carry = 1 and (RA) > 0;

REGISTERS AFFECTED: RA, CS, PI

5.56 Increment memory by a positive integer.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE			
			8	4	4	16
D		INCM N, ADDR	-----			
DX		INCM N, ADDR, RX	A3	N-1	RX	ADDR

DESCRIPTION: The contents of the memory location specified by the Derived Address, DA, is incremented by N, where N is an integer, $1 \leq N \leq 16$. This instruction adds a positive constant to memory. The condition status, CS, is set based on the results of the addition and carry. A fixed point overflow occurs if the operand in memory is positive and the result is negative. The memory location specified is updated to contain the result of the addition process even if a fixed point overflow occurs.

REGISTER TRANSFER DESCRIPTION:

$[DA]^2 \leftarrow [DA]^1 + N$, where $1 \leq N \leq 16$;

$PI_4 \leftarrow 1$, if $[DA]^2 < 0 < [DA]^1$;

(CS) $\leftarrow$ 0010 if carry = 0 and $[DA] = 0$;
 (CS) $\leftarrow$ 0101 if carry = 0 and $[DA] < 0$;
 (CS) $\leftarrow$ 0100 if carry = 0 and $[DA] > 0$;
 (CS) $\leftarrow$ 1010 if carry = 1 and $[DA] = 0$;
 (CS) $\leftarrow$ 1001 if carry = 1 and $[DA] < 0$;
 (CS) $\leftarrow$ 1100 if carry = 1 and $[DA] > 0$;

REGISTERS AFFECTED: CS, PI

MTI-STD-1750A (USAF)
2 July 1980

5.57 Single precision absolute value of register.

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT / OPCODE</u>
			8 4 4
R	ABS	RA, RB	A4 RA RB

DESCRIPTION: If the sign bit of the Derived Operand, DO (i.e., the sign bit of register RB), is a one, its negative or 2's complement is stored into register RA. However, if the sign bit of DO is a zero, it is stored, unchanged, into RA. The condition status, CS, is set based on the result in register RA.

Note: RA may equal RB.

The absolute value of a number with a 1 in the sign bit and all other bits zero is the same word, and causes fixed point overflow to occur.

REGISTER TRANSFER DESCRIPTION:

PI₄ <-- 1, exit, if DO = 8000₁₆;

(RA) <-- |DO|;

(CS) <-- 0001 if (RA) = 8000₁₆;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS, PI

5.58 Double precision absolute value of register.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4

R		DABS RA, RB	A5 RA RB

DESCRIPTION: If the sign bit of the double precision Derived Operand DO (i.e., the sign bit of register (RB, RB+1)), is a one, its negative or 2's complement is stored into register RA and RA+1, such that register RA contains the MSH of the result. However, if the sign bit of DO is a zero, it is stored, unchanged, into RA and RA+1. The condition status, CS, is set based on the result in register RA and RA+1.

Note: RA may equal RB.

The absolute value of a number with a 1 in the sign bit and all other bits zero is the same word, and causes fixed point overflow to occur.

REGISTER TRANSFER DESCRIPTION:

PI₄ <-- 1, exit, if DO = 8000 0000₁₆;

(RA, RA+1) <-- |DO|;

(CS) <-- 0001 if (RA, RA+1) = 8000 0000₁₆;

(CS) <-- 0010 if (RA, RA+1) = 0;

(CS) <-- 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.59 Double precision integer add.

<u>ADDR MODE</u>			<u>FORMAT/OPCODE</u>			
			8	4	4	
R	DAR	RA, RB	A7	RA	RB	
D	DA	RA, ADDR	8	4	4	16
DX	DA	RA, ADDR, RX	A6	RA	RX	ADDR

DESCRIPTION: The double precision Derived Operand (DO) is added to the contents of registers RA and RA+1. The result (a 2's complement 32-bit sum) is stored in registers RA and RA+1. The MSH is in RA. The condition status (CS) is set based on the double precision results in RA and RA+1, and carry. A fixed point overflow occurs if both operands are of the same sign and the sum is of opposite sign.

REGISTER TRANSFER DESCRIPTION:

$(RA, RA+1)^2 \leftarrow (RA, RA+1)^1 + DO;$

$PI_4 \leftarrow 1, \text{ if } (RA_0)^1 = DO_0 \text{ and } (RA_0)^1 \neq (RA_0)^2$

(CS) $\leftarrow$ 0010 if carry = 0 and (RA, RA+1) = 0;
 (CS) $\leftarrow$ 0001 if carry = 0 and (RA, RA+1) < 0;
 (CS) $\leftarrow$ 0100 if carry = 0 and (RA, RA+1) > 0;
 (CS) $\leftarrow$ 1010 if carry = 1 and (RA, RA+1) = 0;
 (CS) $\leftarrow$ 1001 if carry = 1 and (RA, RA+1) < 0;
 (CS) $\leftarrow$ 1100 if carry = 1 and (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, CS, PI

5.60 Floating point add.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
R	FAR	RA, RB	844 A9 RA RB
B	FAB	BR, DSPL	4228 2 0 BR' DSPL $12 \leq BR \leq 16$ $BR' = BR - 12$ $RA = R0$
BX	FABX	BR, RX	42244 4 0 BR' 8 RX $12 \leq BR \leq 16$ $BR' = BR - 12$ $RA = R0$
D	FA	RA, ADDR	84416 A8 RA RX ADDR
DX	FA	RA, ADDR, RX	84416 A8 RA RX ADDR

DESCRIPTION: The floating point Derived Operand, DO, is floating point added to the contents of registers RA and RA + 1. The result is stored in registers RA and RA + 1. The process of this operation is as follows: the mantissa of the number with the smaller algebraic exponent is shifted right and the exponent incremented by one for each bit shifted until the exponents are equal. The mantissas are then added. If the sum overflows the 24-bit mantissa, then the sum is shifted right one position, the sign bit restored, and the exponent incremented by one. If the exponent exceeds $7F_{16}$ as a result of this incrementation, overflow occurs and the operation is terminated. If the sum does not result in exponent overflow, the result is normalized. If in the normalization process the exponent is decremented below 80_{16} , then underflow occurs and a zero is inserted for the result.

MIL-STD-1750A (USAF)
2 July 1980

REGISTER TRANSFER DESCRIPTION:

$n = EA - E0;$

$MO \leftarrow MO$ Shifted Right Arithmetic n positions, if $n > 0$ and $MA \neq 0;$

$MA \leftarrow MA$ Shifted Right Arithmetic $-n$ positions, $EA \leftarrow E0$, if $n < 0$ and $MO \neq 0;$

$MA \leftarrow MA + MO;$

$MA \leftarrow MA$ Shifted Right Arithmetic 1 position, $MA_0 \leftarrow \overline{MA_0}$, $EA \leftarrow EA+1$,
if $OVM = 1;$

$PI_3 \leftarrow 1$, $EA \leftarrow 7F_{16}$, $MA \leftarrow 7FFF\ FF_{16}$, exit, if $EA > 7F_{16}$ and $MA_0 = 0;$

$PI_3 \leftarrow 1$, $EA \leftarrow 7F_{16}$, $MA \leftarrow 8000\ 00_{16}$, exit, if $EA > 7F_{16}$ and $MA_0 = 1;$

$EA, MA \leftarrow$ normalized $EA, MA;$

$PI_6 \leftarrow 1$, $EA \leftarrow 0$, $MA \leftarrow 0$, if $EA < 80_{16};$

$(CS) \leftarrow 0010$ if $(RA, RA+1) = 0;$

$(CS) \leftarrow 0001$ if $(RA, RA+1) < 0;$

$(CS) \leftarrow 0100$ if $(RA, RA+1) > 0;$

REGISTERS AFFECTED: $RA, RA+1, CS, PI$

5.61 Extended precision floating point add.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R		EFAR RA, RB	AB RA RB
			8 4 4 16
D		EFA RA, ADDR	AA RA RX ADDR
DX		EFA RA, ADDR, RX	AA RA RX ADDR

DESCRIPTION: The extended precision floating point Derived Operand, DO, is extended floating point added to the contents of register RA, RA+1, and RA+2. The result is stored in register RA, RA+1, and RA+2. The process of this operation is as follows: the mantissa of the number with the smaller algebraic exponent is shifted right and the exponent is incremented by one for each bit shifted. When the exponents are equal, the mantissas are added. If the sum overflows the 39-bit mantissa, then the sum is shifted right one position, the sign bit restored, and the exponent is incremented by one. If the exponent exceeds $7F_{16}$ as a result of this incrementation, overflow occurs and the operation is terminated. If the sum does not result in exponent overflow, the result is normalized. If in the normalization process the exponent is decremented below 80_{16} , then underflow occurs and a zero is inserted for the result.

REGISTER TRANSFER DESCRIPTION:

$n = EA - DO;$

$MO \leftarrow MO$ Shifted Right Arithmetic n positions, if $n > 0$ and $MA \neq 0;$

$MA \leftarrow MA$ Shifted Right Arithmetic $-n$ positions, $EA \leftarrow EO$, if $n < 0$ and $MO \neq 0;$

$MA \leftarrow MA + MO;$

$MA \leftarrow MA$ Shifted Right Arithmetic 1 position, $MA_0 \leftarrow \overline{MA_0}$, $EA \leftarrow EA+1$,
if $OVM = 1;$

$PI_3 \leftarrow 1$, $EA \leftarrow 7F_{16}$, $MA \leftarrow 7FFF\ FF\ FFFF_{16}$, exit, if $EA > 7F_{16}$ and $MA_0 = 0;$

$PI_3 \leftarrow 1$, $EA \leftarrow 7F_{16}$, $MA \leftarrow 8000\ 00\ 0000_{16}$, exit, if $EA > 7F_{16}$ and $MA_0 = 1;$

$EA, MA \leftarrow$ normalized $EA, MA;$

$PI_6 \leftarrow 1$, $EA \leftarrow 0$, $MA \leftarrow 0$, if $EA < 80_{16};$

$(CS) \leftarrow 0010$ if $(RA, RA+1, RA+2) = 0;$

$(CS) \leftarrow 0001$ if $(RA, RA+1, RA+2) < 0;$

$(CS) \leftarrow 0100$ if $(RA, RA+1, RA+2) > 0;$

REGISTERS AFFECTED: RA, RA+1, RA+2, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.62 Floating point absolute value of register.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE		
			8	4	4
R		FABS RA, RB	AC	RA	RB

DESCRIPTION: If the sign bit of the mantissa of the Derived Operand, DO (i.e., the contents of registers RB and RB+1), is a one, its floating point negative is stored in registers RA and RA+1. The negative of DO is computed by taking the 2's complement of the mantissa and leaving the exponent unchanged. Exceptions to this are negative powers of two: -1.0×2^0 , -1.0×2^1 , ... The absolute value of these are: 0.5×2^1 , 0.5×2^2 , ...; in other words, the DO mantissa is shifted logically right one position and the exponent incremented. A floating point overflow shall occur if DO is the smallest negative number, -1.0×2^{127} . If the sign bit of DO is a zero, it is stored unchanged into RA and RA+1. The condition status, CS, is set based on the result in register RA and RA+1.

Note: RA may equal RB.

DO is assumed to be a normalized number or floating point zero.

REGISTER TRANSFER DESCRIPTION:

EA <-- EA+1, MA <-- 4000 00₁₆, if MO = 8000 00₁₆;

PI₃ <-- 1, EA <-- 7F₁₆, MA <-- 7FFF FF₁₆, exit, if EA > 7F₁₆;

EA <-- EO, MA <-- -MO, if MO < 0, MO ≠ 8000 00₁₆;

EA <-- EO, MA <-- MO, if MO > 0;

(CS) <-- 0010 if (RA, RA+1) = 0;

(CS) <-- 0001 if (RA, RA+1) < 0;

(CS) <-- 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, CS, PI

5.63 Single precision integer subtract.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
R	SR	RA, RB	8 4 4 B1 RA RB
B	SBB	BR, DSPL	4 2 2 8 1 1 BR' DSPL $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
BX	SBBX	BR, RX	4 2 2 4 4 4 0 BR' 5 RX $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
ISP	SISP	RA, N	8 4 4 B2 RA N-1 $1 \leq N \leq 16$
D DX	S S	RA, ADDR RA, ADDR, RX	8 4 4 16 B0 RA RX ADDR
IM	SIM	RA, DATA	8 4 4 16 4A RA 2 DATA

DESCRIPTION: The Derived Operand (DO) is subtracted from the contents of the RA register. The result, a 2's complement difference, is stored in RA. The condition status (CS) is set based on the result in register RA and carry. A fixed point overflow occurs if both operands are of opposite signs and the derived operand is the same as the sign of the difference.

MIL-STD-1750A (USAF)
2 July 1980

REGISTER TRANSFER DESCRIPTION:

$(RA)^2 \leftarrow (RA)^1 - DO$, i.e., $(RA) - DO$ means $\{(RA) + \overline{DO}\} + 1$;

$PI_4 \leftarrow 1$, if $(RA_0)^1 \neq DO_0$ and $(RA_0)^2 = DO_0$

(CS) $\leftarrow$ 0010 if carry = 0 and $(RA) = 0$;

(CS) $\leftarrow$ 0001 if carry = 0 and $(RA) < 0$;

(CS) $\leftarrow$ 0100 if carry = 0 and $(RA) > 0$;

(CS) $\leftarrow$ 1010 if carry = 1 and $(RA) = 0$;

(CS) $\leftarrow$ 1001 if carry = 1 and $(RA) < 0$;

(CS) $\leftarrow$ 1100 if carry = 1 and $(RA) > 0$;

REGISTERS AFFECTED: RA, CS, PI

5.64 Decrement memory by a positive integer.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE			
			8	4	4	16
D		DECM N, ADDR	-----			
DX		DECM N, ADDR, RX	B3	N-1	RX	ADDR

DESCRIPTION: The contents of the memory location specified by the Derived Address, DA, are decremented by N where N is an integer, $1 \leq N \leq 16$. This is the equivalent of a "subtract-from-memory instruction". The condition status, CS, is set based on the results of the subtraction and carry. A fixed point overflow occurs if the operand in memory is negative and the result is positive. The memory location specified is updated to contain the result of the subtraction process even if a fixed point overflow occurs.

REGISTER TRANSFER DESCRIPTION:

$[DA]^2 \leftarrow [DA]^1 - N$, where $1 \leq N \leq 16$;

$PI_4 \leftarrow 1$, if $[DA_0]^1 < 0 < [DA_1]^2$;

(CS) $\leftarrow$ 0010 if carry = 0 and $[DA] = 0$;

(CS) $\leftarrow$ 0001 if carry = 0 and $[DA] < 0$;

(CS) $\leftarrow$ 0100 if carry = 0 and $[DA] > 0$;

(CS) $\leftarrow$ 1010 if carry = 1 and $[DA] = 0$;

(CS) $\leftarrow$ 1001 if carry = 1 and $[DA] < 0$;

(CS) $\leftarrow$ 1100 if carry = 1 and $[DA] > 0$;

REGISTERS AFFECTED: CS, PI

MIL-STD-1750A (USAF)

2 July 1980

5.65 Single precision negate register.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R		NEG RA, RB	B4 RA RB

DESCRIPTION: The negative (i.e., the 2's complement) of the Derived Operand, DO (i.e., the contents of register RB), is stored into register RA. The condition status, CS, is set based on the result in register RA.

Note: The negative of zero is zero.

The negative of a number with a 1 in the sign bit and all other bits zero is the same word, and causes fixed point overflow to occur.

REGISTER TRANSFER DESCRIPTION:

PI₄ <-- 1, exit, if DO = 8000₁₆;

(RA) <-- -DO;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS, PI

5.66 Double precision negate register.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
R		DNEG RA, RB	B5 RA RB

DESCRIPTION: The negative (i.e., the 2's complement) of the Derived Operand, DO (i.e., the contents of register RB and RB + 1), is stored into register RA and RA + 1 such that register RA contains the MSH of the result. The condition status, CS, is set based on the result in register RA and RA + 1.

Note: The negative of zero is zero.

The negative of a number with a 1 in the sign bit and all other bits zero is the same word, and causes fixed point overflow to occur.

REGISTER TRANSFER DESCRIPTION:

$PI_4 \leftarrow 1$, exit, if $DO = 8000\ 0000_{16}$;

$(RA, RA+1) \leftarrow -DO$;

$(CS) \leftarrow 0010$ if $(RA, RA+1) = 0$;

$(CS) \leftarrow 0001$ if $(RA, RA+1) < 0$;

$(CS) \leftarrow 0100$ if $(RA, RA+1) > 0$;

REGISTERS AFFECTED: RA, RA+1, CS, PI

MIL-STD-1750A (USAF)

2 July 1980

5.67 Double precision integer subtract.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R	DSR	RA, RB	B7 RA RB
			8 4 4 16
D	DS	RA, ADDR	B6 RA RX ADDR
DX	DS	RA, ADDR, RX	B6 RA RX ADDR

DESCRIPTION: The double precision Derived Operand, DO, is subtracted from the contents of registers RA and RA + 1. The results, a 2's complement 32-bit difference, is stored in registers RA and RA + 1. The MSH is RA. The condition status (CS) is set based on the double precision results in RA and RA + 1, and carry. A fixed point overflow occurs if both operands are of opposite sign and the derived operand is the same as the sign of the difference.

REGISTER TRANSFER DESCRIPTION:

$(RA, RA+1)^2 \leftarrow (RA, RA+1)^1 - DO$, i.e., $(RA, RA+1) - DO$ means $((RA, RA+1) + \bar{DO}) + 1$;

$PI_4 \leftarrow 1$, if $(RA_0)^1 \neq DO_0$ and $(RA_0)^2 = DO_0$;

(CS) $\leftarrow$ 0010 if carry = 0 and $(RA, RA+1) = 0$;
 (CS) $\leftarrow$ 0001 if carry = 0 and $(RA, RA+1) < 0$;
 (CS) $\leftarrow$ 0100 if carry = 0 and $(RA, RA+1) > 0$;
 (CS) $\leftarrow$ 1010 if carry = 1 and $(RA, RA+1) = 0$;
 (CS) $\leftarrow$ 1001 if carry = 1 and $(RA, RA+1) < 0$;
 (CS) $\leftarrow$ 1100 if carry = 1 and $(RA, RA+1) > 0$;

REGISTERS AFFECTED: RA, RA+1, CS, PI

5.68 Floating point subtract.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE	
			8 4 4	
R		FSR RA, RB	B9 RA RB	
			4 2 2 8	
B		FSB BR, DSPL	2 1 BR' DSPL	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
			4 2 2 4 4	
BX		FSBX BR, RX	4 0 BR' 9 RX	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
			8 4 4 16	
D	FS	RA, ADDR	B8 RA RX ADDR	
DX	FS	RA, ADDR, RX	B8 RA RX ADDR	

DESCRIPTION: The floating point Derived Operand, DO, is floating point subtracted from the contents of registers RA and RA + 1. The result is stored in registers RA and RA + 1. The process of this operation is as follows: the mantissa of the number with the smaller algebraic exponent is shifted right and the exponent incremented by one for each bit shifted until the exponents are equal. The mantissa of the DO is then subtracted from (RA, RA + 1). If the difference overflows the 24-bit mantissa, then it is shifted right one position, the sign bit restored, and the exponent incremented by one. If the exponent exceeds $7F_{16}$ as a result of this incrementation, overflow occurs and the operation is terminated. If the sum does not result in exponent overflow, the result is normalized. If during the normalization process the exponent is decremented below 80_{16} , then underflow occurs and a zero is inserted for the result.

MIL-STD-1750A (USAF)
2 July 1980

REGISTER TRANSFER DESCRIPTION:

$n = EA - E0;$

$MO \leftarrow MO$ Shifted Right Arithmetic n positions, if $n > 0$ and $MA \neq 0;$

$MA \leftarrow MA$ Shifted Right Arithmetic $-n$ positions, $EA \leftarrow E0$, if $n < 0$ and $MO \neq 0;$

$MA \leftarrow MA - MO;$

$MA \leftarrow MA$ Shifted Right Arithmetic 1 position, $MA_0 \leftarrow \overline{MA_0}$, $EA \leftarrow EA+1$,
if $OVM = 1;$

$PI_3 \leftarrow 1$, $EA \leftarrow 7F_{16}$, $MA \leftarrow 7FFF\ FF_{16}$, exit, if $EA > 7F_{16}$ and $MA_0 = 0;$

$PI_3 \leftarrow 1$, $EA \leftarrow 7F_{16}$, $MA \leftarrow 8000\ 00_{16}$, exit, if $EA > 7F_{16}$ and $MA_0 = 1;$

$EA, MA \leftarrow$ normalized $EA, MA;$

$PI_6 \leftarrow 1$, $EA \leftarrow 0$, $MA \leftarrow 0$, if $EA < 80_{16};$

(CS) $\leftarrow 0010$ if $(RA, RA+1) = 0;$

(CS) $\leftarrow 0001$ if $(RA, RA+1) < 0;$

(CS) $\leftarrow 0100$ if $(RA, RA+1) > 0;$

REGISTERS AFFECTED: RA, RA+1, CS, PI

5.69 Extended precision floating point subtract.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
R		EFSR RA, RB	BB RA RB
			8 4 4 16
D	EFS	RA, ADDR	BA RA RX ADDR
DX	EFS	RA, ADDR, RX	BA RA RX ADDR

DESCRIPTION: The extended precision floating point Derived Operand, DO, is extended floating point subtracted from the contents of registers RA, RA + 1, and RA + 2. The result is stored in registers RA, RA + 1, and RA + 2. The process of this operation is as follows: the mantissa of the number with the smaller algebraic exponent is shifted right and the exponent is incremented by one for each bit shifted. When the exponents are equal, the mantissas are subtracted. If the difference overflows the 39-bit mantissa, then the difference is shifted right one position, the sign bit restored, and the exponent is incremented. If the exponent exceeds $7F_{16}$ as a result of this incrementation, overflow occurs and the operation is terminated. If the sum does not result in exponent overflow, the result is normalized. If during the normalization process the exponent is decremented below 80_{16} , then underflow occurs and a zero is inserted for the result.

REGISTER TRANSFER DESCRIPTION:

$n = EA - E0;$

$MO \leftarrow MO$ Shifted Right Arithmetic n positions, if $n > 0$ and $MA \neq 0;$

$MA \leftarrow MA$ Shifted Right Arithmetic $-n$ positions, $EA \leftarrow E0$, if $n < 0$ and $MO \neq 0;$

$MA \leftarrow MA - MO;$

$MA \leftarrow MA$ Shifted Right Arithmetic 1 position, $MA_0 \leftarrow \overline{MA_0}$, $EA \leftarrow EA + 1$,
if $OVM = 1;$

$PI_3 \leftarrow 1$, $EA \leftarrow 7F_{16}$, $MA \leftarrow 7FFF\ FF\ FFFF_{16}$, exit, if $EA > 7F_{16}$ and $MA_0 = 0;$

$PI_3 \leftarrow 1$, $EA \leftarrow 7F_{16}$, $MA \leftarrow 8000\ 00\ 0000_{16}$, exit, if $EA > 7F_{16}$ and $MA_0 = 1;$

$EA, MA \leftarrow$ normalized $EA, MA;$

$PI_6 \leftarrow 1$, $EA \leftarrow 0$, $MA \leftarrow 0$, if $EA < 80_{16};$

$(CS) \leftarrow 0010$ if $(RA, RA+1, RA+2) = 0;$

$(CS) \leftarrow 0001$ if $(RA, RA+1, RA+2) < 0;$

$(CS) \leftarrow 0100$ if $(RA, RA+1, RA+2) > 0;$

REGISTERS AFFECTED: RA, RA+1, RA+2, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.70 Floating point negate register.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE		
			8	4	4
R		FNEG RA, RB	BC	RA	RB

DESCRIPTION: The 24-bit mantissa of the Derived Operand, DO, i.e., the floating point number in registers RB and RB + 1, is 2's complemented. The exponent remains unchanged. The result, the negative of the original number, is stored in RA and RA + 1. The 2's complement of a floating point zero is a floating point zero. Exceptions to this are all powers of two: -1.0×2^n and $0.5 \times 2^{n+1}$; i.e., when the mantissa is either $8000\ 00_{16}$ or $4000\ 00_{16}$. The negation of 0.5×2^n is $-1.0 \times 2^{n-1}$; i.e., the mantissa is shifted left one position and the exponent decremented by one. Conversely, the negation of -1.0×2^n is $0.5 \times 2^{n+1}$; i.e., the mantissa is shifted right one position and the exponent is incremented by one. A floating point overflow occurs for the negation of the smallest negative number, -1.0×2^{127} . A floating point underflow occurs for the negation of the smallest positive number, 0.5×2^{-128} , and causes the result to be zero. The condition status, CS, is set based on the result in registers RA and RA + 1.

Note: RA may equal RB.

REGISTER TRANSFER DESCRIPTION:

$PI_3 \leftarrow 1$, $EA \leftarrow 7F_{15}$, $MO \leftarrow 7FFF\ FF_{16}$, exit, if $DO = 8000\ 007F_{16}$;

$PI_6 \leftarrow 1$, $EA \leftarrow 0$, $MA \leftarrow 0$, exit, if $DO = 4000\ 0080_{16}$;

$EA \leftarrow EO+1$, $MA \leftarrow 4000\ 00_{16}$, if $MO = 8000\ 00_{16}$;

$EA \leftarrow EO-1$, $MA \leftarrow 8000\ 00_{16}$, if $MO = 4000\ 00_{16}$;

$EA \leftarrow EO$, $MA \leftarrow -MO$, if $MO \neq 8000\ 00_{16}$ or $4000\ 00_{16}$;

(CS) $\leftarrow 0010$ if $(RA, RA+1) = 0$;

(CS) $\leftarrow 0001$ if $(RA, RA+1) < 0$;

(CS) $\leftarrow 0100$ if $(RA, RA+1) > 0$;

REGISTERS AFFECTED: RA, RA+1, CS, PI

5.71 Single precision integer multiply with 16-bit product.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
R		MSR RA, RB	844 ----- C1 RA RB -----
ISP		MISP RA, N	844 ----- C2 RA N-1 ----- $1 \leq N \leq 16$
ISN		MISN RA, N	844 ----- C3 RA N-1 ----- $1 \leq N \leq 16$
D DX	MS MS	RA, ADDR RA, ADDR, RX	84416 ----- C0 RA RX ADDR -----
IM		MSIM RA, DATA	84416 ----- 4A RA 4 DATA -----

DESCRIPTION: The Derived Operand, DO, is multiplied by the contents of register RA. The LSH of the result, a 16-bit, 2's complement integer, is stored in register RA. The Condition Status, CS, is set based on the result in register RA. A fixed point overflow occurs if (1) both operands are of the same sign and the MSH of the product is not zero, or the sign bit of the LSH is not zero, or (2) if the operands are of opposite sign and the MSH of the product is not FFFF₁₆, or the sign bit of the LSH is not one. A fixed point overflow does not occur if either of the operands is zero.

REGISTER TRANSFER DESCRIPTION:

$(RQ, RQ+1) \leftarrow (RA)^1 \times DO;$

$(RA)^2 \leftarrow (RQ+1);$

$PI_4 \leftarrow 1, \text{ if } \{(RA_0)^1 = DO_0 \text{ and } \{(RQ) \neq 0 \text{ or } (RQ+1_0) = 1\} \text{ or } \{(RA_0)^1 \neq DO_0 \text{ and } \{(RQ) \neq FFFF_{16} \text{ or } (RQ+1_0) = 0\} \text{ and } \{(RA)^1 \neq 0 \text{ and } DO \neq 0\}\};$

$(CS) \leftarrow 0010 \text{ if } (RA) = 0;$

$(CS) \leftarrow 0001 \text{ if } (RA) < 0;$

$(CS) \leftarrow 0100 \text{ if } (RA) > 0;$

REGISTERS AFFECTED: RA, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.72 Single precision integer multiply with 32-bit product.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4 ----- C5 RA RB -----
R	MR	RA, RB	
			4 2 2 8 ----- 1 2 BR' DSPL -----
B	MB	BR, DSPL	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
			4 2 2 4 4 ----- 4 0 BR' 6 RX -----
BX	MBX	BR, RX	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
			8 4 4 16 ----- C4 RA RX ADDR -----
D	M	RA, ADDR	
DX	M	RA, ADDR, RX	
			8 4 4 16 ----- 4A RA 3 DATA -----
IM	MIM	RA, DATA	

DESCRIPTION: The Derived Operand, DO, is multiplied by the contents of register RA. The result, a 32-bit, 2's complement integer, is stored in registers RA and RA + 1 with the MSH of the product in register RA. The Condition Status, CS, is set based on the result in registers RA and RA + 1.

SPECIAL CASE: DO = (RA) = 8000 (the largest negative number), then DO x (RA) = 4000 0000.

REGISTER TRANSFER DESCRIPTION:

(RA, RA+1) <-- (RA) x DO;

(CS) <-- 0010 if (RA, RA+1) = 0;

(CS) <-- 0001 if (RA, RA+1) < 0;

(CS) <-- 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, CS

5.73 Double precision integer multiply.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
R	DMR	RA, RB	C7 RA RB
			8 4 4 16
D	DM	RA, ADDR	C6 RA RX ADDR
DX	DM	RA, ADDR, RX	C6 RA RX ADDR

DESCRIPTION: The double precision Derived Operand, DO, a 32-bit 2's complement number, is multiplied by the contents of registers RA and RA + 1, a 32-bit 2's complement number, with the MSH in RA. The LSH of the product is retained in RA and RA + 1 as a 32-bit, 2's complement number. The MSH is lost. The Condition Status, CS, is set based on the double precision result in registers RA and RA + 1. A fixed point overflow occurs if (1) both operands are of the same sign and the MSH of the product is not zero, or the sign bit of the LSH is not zero, or (2) if the operands are of opposite sign and the MSH of the product is not FFFF FFFF₁₆, or the sign bit of the LSH is not one. A fixed point overflow does not occur if either of the operands is zero.

REGISTER TRANSFER DESCRIPTION:

(RQ, RQ+1, RQ+2, RQ+3) <-- (RA, RA+1)¹ x DO;

(RA, RA+1)² <-- (RQ+2, RQ+3);

PI₄ <-- 1, if {(RA₀)¹ = DO₀ and {(RQ, RQ+1) ≠ 0 or (RQ+2₀) = 1}} or
{(RA₀)¹ ≠ DO₀ and {(RQ, RQ+1) ≠ FFFF FFFF₁₆ or (RQ+2₀) = 0} and
{(RA)¹ ≠ 0 and DO ≠ 0}};

(CS) <-- 0010 if (RA, RA+1) = 0;

(CS) <-- 0001 if (RA, RA+1) < 0;

(CS) <-- 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.74 Floating point multiply.

<u>ADDR</u>		<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
				8	4	4	
R	FMR	RA, RB		C9	RA	RB	
				4	2	2	8
B	FMB	BR, DSPL		2	2	BR'	DSPL
							12 ≤ BR ≤ 15 BR' = BR - 12 RA = R0
				4	2	2	4
BX	FMBX	BR, RX		4	0	BR'	A
							12 ≤ BR ≤ 15 BR' = BR - 12 RA = R0
				8	4	4	16
D	FM	RA, ADDR		C8	RA	RX	
DX	FM	RA, ADDR, RX					ADDR

DESCRIPTION: The floating point Derived Operand, DO, is floating point multiplied by the contents of register RA and RA + 1. The result is stored in register RA and RA + 1. The process of the operation is as follows: the exponents of the operands are added. If the sum exceeds $7F_{16}$, a floating point overflow occurs. If the sum is less than 80_{16} , then underflow occurs and the result set to zero. The operand mantissas are multiplied and the result normalized and stored in RA and RA + 1. An exceptional case is when both operands are negative powers of two: $(-1.0 \times 2^n) \times (-1.0 \times 2^m)$; the result is a $0.5 \times 2^{n+m+1}$. If $n+m \approx 7F_{16}$, this shall yield an exponent overflow, floating point overflow occurs. Also, it is possible that the normalization process may yield an exponent underflow; if this occurs, then the result is forced to zero. The condition status, CS, is set based on the result in RA and RA + 1.

REGISTER TRANSFER DESCRIPTION:

$n = EA + EO;$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 7FFF\ FF_{16}, \text{exit, if } n > 7F_{16} \text{ and } MA_0 = MO_0;$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 8000\ 00_{16}, \text{exit, if } n > 7F_{16} \text{ and } MA_0 \neq MO_0;$

$PI_6 \leftarrow 1, EA \leftarrow 0, MA \leftarrow 0, \text{exit, if } n < 80_{16};$

$MP \leftarrow MA \times MO; (\text{integer multiply})$

$MP \leftarrow MP \text{ shift left 1 position};$

$n \leftarrow n + 1, MP_{0-23} \leftarrow 4000\ 00_{16}, \text{if } MP_{0-23} = 8000\ 00_{16};$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 7FFF\ FF_{16}, \text{exit, if } n > 7F_{16} \text{ and } MP_0 = 0;$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 8000\ 00_{16}, \text{exit, if } n > 7F_{16} \text{ and } MP_0 = 1;$

$n, MP \leftarrow \text{normalized } n, MP;$

$PI_6 \leftarrow 1, EA \leftarrow 0, MA \leftarrow 0, \text{exit, if } n < 80_{16};$

$EA \leftarrow n;$

$MA \leftarrow MP_{0-23};$

$(CS) \leftarrow 0010 \quad \text{if } (RA, RA+1) = 0;$

$(CS) \leftarrow 0001 \quad \text{if } (RA, RA+1) < 0;$

$(CS) \leftarrow 0100 \quad \text{if } (RA, RA+1) > 0;$

REGISTERS AFFECTED: RA, RA+1, CS, PI

MIL-STD-1750A (USAF)

2 July 1980

5.75. Extended precision floating point multiply.

ADDR	MODE	MNEMONIC	FORMAT/CPCODE
			8 4
R		EFMR RA, RB	[CB] [RA] [RB]
			8 4 4 16
D		EFM RA, ADDR	[CA] [RA] [RX] [ADDR]
DX		EFM RA, ADDR, RX	

DESCRIPTION: The extended precision floating Derived Operand, DX, is extended floating point multiplied by the contents of registers RA, RA + 1, and RA + 2. The result is stored in registers RA, RA + 1, and RA + 2. The process of the operation is as follows: the exponent of the operands are added. If the sum exceeds $7F_{16}$, a floating point overflow occurs. If the sum is less than 80_{16} , then underflow occurs and the result set to zero. The operand mantissas are multiplied and the result normalized and stored in RA, RA + 1, and RA + 2. The condition status, CS, is set based on the result in RA, RA + 1, and RA + 2.

REGISTER TRANSFER DESCRIPTION:

$n = EA + E0;$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 7FFF\ FF\ FFFF_{16},$ exit, if $n > 7F_{16}$ and $MA_0 = MO_0;$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 8000\ 00\ 0000_{16},$ exit, if $n > 7F_{16}$ and $MA_0 \neq MO_0;$

$PI_6 \leftarrow 1, EA \leftarrow 0, MA \leftarrow 0,$ exit, if $n < 80_{16};$

$MP \leftarrow MA \times MO;$ (integer multiply)

$MP \leftarrow MP$ shift left 1 position;

$n \leftarrow n + 1, MP_{0-39} \leftarrow 4000\ 00\ 0000_{16},$ if $MP_{0-39} = 8000\ 00\ 0000_{16};$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 7FFF\ FF\ FFFF_{16},$ exit, if $n > 7F_{16}$ and $MP_0 = 0;$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 8000\ 00\ 0000_{16},$ exit, if $n > 7F_{16}$ and $MP_0 = 1;$

$n, MP \leftarrow$ normalized $n, MP;$

$PI_6 \leftarrow 1, EA \leftarrow 0, MA \leftarrow 0,$ if $n < 80_{16};$

$EA \leftarrow n;$

$MA \leftarrow MP_{0-39};$

(CS) $\leftarrow 0010$ if $(RA, RA+1, RA+2) = 0;$

(CS) $\leftarrow 0001$ if $(RA, RA+1, RA+2) < 0;$

(CS) $\leftarrow 0100$ if $(RA, RA+1, RA+2) > 0;$

REGISTERS AFFECTED: RA, RA+1, RA+2, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.76 Single precision integer divide with 16-bit dividend.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
			8 4 4
R	DVR	RA, RB	D1 RA RB
			8 4 4
ISP	DISP	RA, N	D2 RA N-1 $1 \leq N \leq 16$
			8 4 4
ISN	DISN	RA, N	D3 RA N-1 $1 \leq N \leq 16$
			8 4 4 16
D	DV	RA, ADDR	D0 RA RX ADDR
DX	DV	RA, ADDR, RX	D0 RA RX ADDR
			8 4 4 16
IM	DVIM	RA, DATA	4A RA 6 DATA

DESCRIPTION: The contents of register RA are divided by the Derived Operand, DO, a single precision, 2's complement number. The result is stored in registers RA and RA + 1 such that RA stores the single precision integer quotient and RA + 1 stores the remainder. The Condition Status, CS, is set based on the result in RA. A fixed point overflow occurs if the divisor, DO, is zero, or if the dividend is 8000_{16} and the divisor is $FFFF_{16}$.

Note: The sign of the non-zero remainder is the same as the sign of the dividend.

REGISTER TRANSFER DESCRIPTION:

$\{RA, RA+1\} \leftarrow (RA) / DO;$

$PI_4 \leftarrow 1$, if $DO = 0$ or $\{RA = 8000_{16} \text{ and } DO = FFFF_{16}\};$

$(CS) \leftarrow 0010$ if $(RA) = 0;$

$(CS) \leftarrow 0001$ if $(RA) < 0;$

$(CS) \leftarrow 0100$ if $(RA) > 0;$

REGISTERS AFFECTED: RA, RA+1, CS, PI

5.77 Single precision integer divide with 32-bit dividend.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
R	DR	RA, RB	844 D5 RA RB
B	DB	BR, DSPL	4228 1 3 BR' DSPL $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
BX	DBX	BR, RX	42244 4 0 BR' 7 RX $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
D DX	D D	RA, ADDR RA, ADDR, RX	84416 D4 RA RX ADDR
IM	DIM	RA, DATA	84416 4A RA 5 DATA

DESCRIPTION: The contents of registers RA and RA + 1, a double precision 2's complement number, are divided by the Derived Operand, DO, a single precision, 2's complement number. RA contains the MSH of the 32-bit dividend. The result is stored in registers RA and RA + 1 such that RA stores the single precision integer quotient and RA + 1 stores the remainder. The Condition Status, CS, is set based on the result in RA. A fixed point overflow occurs if the divisor equals zero or if the magnitude of the MSH of the dividend is equal to or greater than the magnitude of the divisor (i.e., the quotient exceeds 15 bits).

Note: The sign of the non-zero remainder is the same as that of the dividend.

REGISTER TRANSFER DESCRIPTION:

(RA, RA+1) <-- (RA, RA+1) / DO;

PI₄ <-- 1, if DO = 0 or |(RA)| ≥ |DO|;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, RA+1, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.78 Double precision integer divide

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	
R	DDR	RA, RB	D7	RA	RB	
			8	4	4	16
D	DD	RA, ADDR	D6	RA	RX	ADDR
DX	DD	RA, ADDR, RX	D6	RA	RX	ADDR

DESCRIPTION: The contents of registers RA and RA + 1, a double precision 2's complement number, are divided by the Derived Operand, DX, a double precision 2's complement number. RA contains the MSH of the 32-bit dividend. The quotient part of the integer result is stored in registers RA and RA + 1 (with the MSH in RA) and the remainder is lost. The Condition Status, CS, is set based on the results in registers RA and RA + 1. A fixed point overflow occurs if the divisor, DX, is zero, or if the dividend is 8000₁₆ and the divisor is FFFF₁₆.

REGISTER TRANSFER DESCRIPTION:

(RA, RA+1) <-- (RA, RA+1) / DX;

PI₄ <-- 1, if DX = 0 or {RA = 8000₁₆ and DX = FFFF₁₆};

(CS) <-- 0010 if (RA, RA+1) = 0;

(CS) <-- 0001 if (RA, RA+1) < 0;

(CS) <-- 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, CS, PI

5.79 Floating point divide.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE	
R	FDR	RA, RB	844 D9 RA RB	
B	FDB	BR, DSPL	4228 2 3 BR' DSPL	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
BX	FDBX	BR, RX	42244 4 0 BR' B RX	$12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
D DX	FD FD	RA, ADDR RA, ADDR, RX	84416 D8 RA RX ADDR	

DESCRIPTION: The floating point number in registers RA and RA + 1 is divided by the floating point Derived Operand, DO. The result is stored in register RA and RA + 1. A floating point overflow occurs if the exponent result exceeds $7F_{16}$ at any point in the calculation process. Underflow occurs if the exponent result is less than 80_{16} at any point in the process. If underflow occurs, then the quotient is forced to zero. A divide by zero yields a floating point overflow.

MIL-STD-1750A (USAF)
2 July 1980

REGISTER TRANSFER DESCRIPTION:

$n = EA - E0;$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 7FFF FF_{16}, \text{exit,}$
if $MA_0 = MQ_0$ and $\{n > 7F_{16} \text{ or } DO = 0\};$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 8000 00_{16}, \text{exit,}$
if $MA_0 \neq MQ_0$ and $\{n > 7F_{16} \text{ or } DO = 0\};$

$PI_6 \leftarrow 1, EA \leftarrow 0, MA \leftarrow 0, \text{exit, if } n < 80_{16};$

$MQ \leftarrow MA / MQ;$

$MQ \leftarrow MQ \text{ Shift Right Arithmetic 1 position, } n \leftarrow n + 1, \text{ if } |MQ| \geq 1.0;$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 7FFF FF_{16}, \text{exit, if } n > 7F_{16} \text{ and } MQ_0 = 0;$

$PI_3 \leftarrow 1, EA \leftarrow 7F_{16}, MA \leftarrow 8000 00_{16}, \text{exit, if } n > 7F_{16} \text{ and } MQ_0 = 1;$

$EA \leftarrow n;$

$MA \leftarrow MQ_{0-23};$

$(CS) \leftarrow 0010 \text{ if } (RA, RA+1) = 0;$

$(CS) \leftarrow 0001 \text{ if } (RA, RA+1) < 0;$

$(CS) \leftarrow 0100 \text{ if } (RA, RA+1) > 0;$

REGISTERS AFFECTED: RA, RA+1, CS, PI

5.80 Extended precision floating point divide

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			844 ----- DB RA RB -----
R		E+DR RA,RB	
			84416 ----- DA RA RX ADDR -----
D		EFD RA,ADDR	
DX		EFD RA,ADDR,RX	

DESCRIPTION: The contents of registers RA, RA + 1, and RA + 2 are extended precision floating point divided by the extended precision floating point Derived Operand, DO. The result is stored in register RA, RA + 1, and RA + 2. A floating point overflow occurs if the exponent result exceeds $7F_{16}$ at any point in the calculation process. Underflow occurs if the exponent result is less than 80_{16} at any point in the process. If underflow occurs, then the quotient is forced to zero. A divide by zero yields a floating point overflow.

REGISTER TRANSFER DESCRIPTION:

n ← EA EO;

PI₃ ← 1, EA ← $7F_{16}$, MA ← $7FFF\ FF\ FFFF_{16}$, exit,
if MA₀ = MO₀ and {n > $7F_{16}$ or DO = 0};

PI₃ ← 1, EA ← $7F_{16}$, MA ← $8000\ 00\ 0000_{16}$, exit,
if MA₀ ≠ MO₀ and {n > $7F_{16}$ or DO = 0};

PI₆ ← 1, EA ← 0, MA ← 0, exit, if n < 80_{16} ;

MQ ← MA / MO;

MQ ← MQ Shift Right Arithmetic 1 position, n ← n + 1, if |MQ| ≥ 1.0;

PI₃ ← 1, EA ← $7F_{16}$, MA ← $7FFF\ FF\ FFFF_{16}$, exit, if n > $7F_{16}$ and MQ₀ = 0;

PI₃ ← 1, EA ← $7F_{16}$, MA ← $8000\ 00\ 0000_{16}$, exit, if n > $7F_{16}$ and MQ₀ = 1;

EA ← n;

MA ← MQ₀₋₃₉;

(CS) ← 0010 if (RA, RA+1, RA+2) = 0;

(CS) ← 0001 if (RA, RA+1, RA+2) < 0;

(CS) ← 0100 if (RA, RA+1, RA+2) > 0;

REGISTERS AFFECTED: RA, RA+1, RA+2, CS, PI

MIL-STD-1750A (USAF)
2 July 1980

5.81 Inclusive logical OR.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
R	ORR	RA, RB	844 E1 RA RB
B	ORB	BR, DSPL	4228 3 0 BR' DSPL $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
BX	ORBX	BR, RX	42244 4 0 BR' F RX $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
D DX	OR OR	RA, ADDR RA, ADDR, RX	84416 E0 RA RX ADDR
IM	ORIM	RA, DATA	84416 4A RA B DATA

DESCRIPTION: The Derived Operand, DO, is bit-by-bit inclusively ORed with the contents of RA. The result is stored in register RA. The condition status, CS, is set based on the result in register RA.

REGISTER TRANSFER DESCRIPTION:

(RA) <-- (RA) v DO;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS

582 Logical AND

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
R	ANDR	RA, RB	8 4 4 E3 RA RB
B	ANDB	BR, DSPL	4 2 2 8 3 1 BR' DSPL $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
BX	ANDX	BR, RX	4 2 2 4 4 4 0 BR' E RX $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
D DX	AND AND	RA, ADDR RA, ADDR, RX	8 4 4 16 E2 RA RX ADDR
IM	ANDM	RA, DATA	8 4 4 16 4A RA 7 DATA

DESCRIPTION: The Derived Operand, DO, is bit-by-bit ANDed with the contents of register RA. The result is stored in register RA. The condition status, CS, is set based on the result in register RA.

REGISTER TRANSFER DESCRIPTION:

(RA) <-- (RA) + DO;

(CS) <-- 0010 if (RA) = 0;
(CS) <-- 0001 if (RA) < 0;
(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS

MIL-STD-1750A (USAF)
2 July 1980

5.83 Exclusive logical OR.

<u>ADDR MODE</u>			<u>FORMAT/OPCODE</u>			
			8	4	4	
R	XORR	RA, RB	E5	RA	RB	
D	XOR	RA, ADDR	8	4	4	16
DX	XOR	RA, ADDR, RX	E4	RA	RX	ADDR
IM	XORM	RA, DATA	8	4	4	16
			4A	RA	9	DATA

DESCRIPTION: The Derived Operand, DO, is bit-by-bit exclusively ORed with the contents of RA. The result is stored in RA. The condition status, CS, is set based on the result in RA.

REGISTER TRANSFER DESCRIPTION:

(RA) <-- (RA) $\oplus$ DO;
(CS) <-- 0010 if (RA) = 0;
(CS) <-- 0001 if (RA) < 0;
(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS

5.84 Logical NAND.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4 ----- E7 RA RB -----
R	NR	RA, RB	
			8 4 4 16 ----- E6 RA RX ADDR -----
D	N	RA, ADDR	
DX	N	RA, ADDR, RX	
			8 4 4 16 ----- 4A RA B DATA -----
IM	NIM	RA, DATA	

DESCRIPTION: The Derived Operand, DO, is bit-by-bit logically NAnDed with the contents of register RA. The result is stored in RA.

Note: The logical NOT of a register can be attained with a NR instruction with RA = RB.

REGISTER TRANSFER DESCRIPTION:

(RA) <-- $\overline{(RA)} \uparrow DO$;

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS

MIL-STD-1750A (USAF)
2 July 1980

5.85 Convert floating point to 16-bit integer.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R	FIX	RA, RB	E8 RA RB

DESCRIPTION: The integer portion of the floating point Derived Operand, DO (i.e., the contents of registers RB and RB+1), is stored into register RA. If the actual value of the DO floating point exponent is greater than $0F_{16}$, then RA remains unchanged and a fixed point overflow occurs. The condition status, CS, is set based on the result in RA.

Note: The algorithm truncates toward zero.

REGISTER TRANSFER DESCRIPTION:

$PI_4 \leftarrow 1$, exit, if $EO > 0F_{16}$;

(RA) $\leftarrow$ Integer portion of DO;

(CS) $\leftarrow$ 0010 if (RA) = 0;

(CS) $\leftarrow$ 0001 if (RA) < 0;

(CS) $\leftarrow$ 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, CS, PI

5.86 Convert 16-bit integer to floating point.

<u>ADDR MODE</u>		<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>		
			8	4	4
R	FLT	RA, RB	E9	RA	RB

DESCRIPTION: The integer Derived Operand, DO (i.e., the contents of register RB), is converted to Single Precision floating point format and stored in register RA and RA + 1. The condition status, CS, is set based on the results in RA and RA + 1. The operation process is as follows: The exponent is initially considered to be $0F_{16}$. The integer value in RB is normalized, i.e., the number is left shifted and the exponent decremented for each shift until the sign bit and the next MSB are unequal, and the exponent and mantissa stored in the proper fields of RA and RA + 1.

Note: RA may equal RB.

An integer zero, 0000_{16} , is converted to a floating point zero, $0000\ 0000_{16}$.

REGISTER TRANSFER DESCRIPTION:

EA $\leftarrow$ 0, MA $\leftarrow$ 0, exit, if (RB) = 0;

EA $\leftarrow$ $0F_{16}$;

MA $\leftarrow$ (RB);

EA, MA $\leftarrow$ normalize EA, MA;

(CS) $\leftarrow$ 0010 if (RA, RA+1) = 0;

(CS) $\leftarrow$ 0001 if (RA, RA+1) < 0;

(CS) $\leftarrow$ 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: RA, RA+1, CS

MIL-STD-1750A (USAF)
2 July 1980

5.87 Convert extended precision floating point to 32-bit integer.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R		EFIX RA,RB	----- EA RA RB -----

DESCRIPTION: The integer portion of the floating point Derived Operand, DO (i.e., the contents of registers RB, RB+1, and RB+2), is stored into register RA and RA+1. If the actual value of the DO floating point exponent is greater than $1F_{16}$, then RA and RA+1 remain unchanged and a fixed point overflow occurs. The condition status, CS, is set based on the result in RA and RA+1.

Note: The algorithm truncates toward zero.

REGISTER TRANSFER DESCRIPTION:

$PI_4 \leftarrow 1$, exit, if $EO > 1F_{16}$;

$(RA, RA+1) \leftarrow$ Integer portion of DO;

(CS) $\leftarrow$ 0010 if $(RA, RA+1) = 0$;

(CS) $\leftarrow$ 0001 if $(RA, RA+1) < 0$;

(CS) $\leftarrow$ 0100 if $(RA, RA+1) > 0$;

REGISTERS AFFECTED: RA, RA+1, CS, PI

588 Convert 32-bit integer to extended precision floating point.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R		EFLT RA, RB	EB RA RB

DESCRIPTION: The double precision integer Derived Operand, DO (i.e., the contents of registers RB and RB+1), is converted to Extended Precision floating point format and stored in register RA, RA+1, and RA+2. The condition status, CS, is set based on the result in RA, RA+1, and RA+2. The operation process is as follows: The exponent is initially considered to be $1F_{16}$. The integer value in RB, RB+1 is normalized, i.e., the number is left shifted and the exponent decremented for each shift until the sign bit and the next MSB are unequal, and the exponent and mantissa stored in the proper field of RA, RA+1, and RA+2.

Note: RA may equal RB.

An integer zero, $0000\ 0000_{16}$, is converted to an extended floating point zero, $0000\ 0000\ 0000_{16}$.

REGISTER TRANSFER DESCRIPTION:

EA $\leftarrow$ 0, MA $\leftarrow$ 0, exit, if (RB, RB+1) = 0;

EA $\leftarrow$ $1F_{16}$, MA $\leftarrow$ (RB, RB+1);

EA, MA $\leftarrow$ normalized EA, MA;

(CS) $\leftarrow$ 0010 if (RA, RA+1, RA+2) = 0;

(CS) $\leftarrow$ 0001 if (RA, RA+1, RA+2) < 0;

(CS) $\leftarrow$ 0100 if (RA, RA+1, RA+2) > 0;

REGISTERS AFFECTED: RA, RA+1, RA+2, CS

MIT-STD-1750A (USAF)

2 July 1980

5.89 Exchange bytes in register

ADDR MODE MNEMONIC

FORMAT/OPCODE

S XBR RA

	8	4	4
	EC	RA	0

DESCRIPTION: The upper byte of register RA is exchanged with the lower byte of register RA. The CS is set based on the result in register RA.

REGISTER TRANSFER DESCRIPTION:

$(RA)_{0-7} \leftrightarrow (RA)_{8-15}$

$(CS) \leftarrow 0010$ if $(RA) = 0$;

$(CS) \leftarrow 0001$ if $(RA) < 0$;

$(CS) \leftarrow 0100$ if $(RA) > 0$;

REGISTERS AFFECTED: RA, CS

590 Exchange words in registers.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>
			8 4 4
R	XWR	RA, RB	ED RA RB

DESCRIPTION: The contents of register RA are exchanged with the contents of register RB. The CS is set based on the result in register RA

REGISTER TRANSFER DESCRIPTION:

(RA) <--> (RB);

(CS) <-- 0010 if (RA) = 0;

(CS) <-- 0001 if (RA) < 0;

(CS) <-- 0100 if (RA) > 0;

REGISTERS AFFECTED: RA, RB, CS

MIL-STD-1750A (USAF)
2 July 1980

5.91 Single precision compare.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
R	CR	RA, RB	844 ----- F1 RA RB -----
B	CB	BR, DSPL	4228 ----- 3 2 BR' DSPL ----- $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
BX	CBX	BR, RX	42244 ----- 4 0 BR' C RX ----- $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R2$
ISP	CISP	RA, N	844 ----- F2 RA N-1 ----- $1 \leq N \leq 16$
ISN	CISN	RA, N	844 ----- F3 RA N-1 ----- $1 \leq N \leq 16$
D DX	C C	RA, ADDR RA, ADDR, RX	84416 ----- F0 RA RX ADDR -----
IM	CIM	RA, DATA	84416 ----- 4A RA A DATA -----

DESCRIPTION: The single precision Derived Operand, DO, is compared to the contents of RA. Then, the Condition Status, CS, is set based on whether the contents of RA is less than, equal to, or greater than the DO. The contents of RA are unchanged.

REGISTER TRANSFER DESCRIPTION:

(RA) : DO;

(CS) <-- 0010 if (RA) = DO;

(CS) <-- 0001 if (RA) < DO;

(CS) <-- 0100 if (RA) > DO;

REGISTERS AFFECTED: CS

CR,CB,CBX,CISP,CISN,C,CIM

132

364

5.92 Compare between limits.

<u>ADDR</u> <u>MODE</u>		<u>MNEMONIC</u>	<u>FORMAT / OPCODE</u>			
			8	4	4	16
D	CBL	RA, ADDR	-----			
DX	CBL	RA, ADDR, RX	F4	RA	RX	ADDR

DESCRIPTION: The contents of register RA are compared to two different sixteen bit derived operands, DO1 and DO2. The derived operands, DO1 and DO2 are located at DA and DA + 1, respectively, and their values are defined such that $DO1 \leq DO2$. The CS is set based on the results. If the values for DO1 and DO2 are defined incorrectly (that is, $DO1 > DO2$), then CS is set to 1000

REGISTER TRANSFER DESCRIPTION:

(CS) <-- 1000 if $DO1 > DO2$, exit;
(CS) <-- 0001 if $(RA) < DO1$;
(CS) <-- 0010 if $DO1 \leq (RA) \leq DO2$;
(CS) <-- 0100 if $(RA) > DO2$;

REGISTERS AFFECTED: CS

MIL-STD-1750A (USAF)

2 July 1980

5.93 Double precision compare.

<u>ADDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	
R	DCR	RA, RB	F7	RA	RB	
D	DC	RA, ADDR	8	4	4	16
DX	DC	RA, ADDR, RX	F6	RA	RX	ADDR

DESCRIPTION: The double precision Derived Operand, DO, is compared to the contents of registers RA and RA+1 where RA contains the MSH of a double precision word. Then, the Condition Status, CS, 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 and RA+1 are unchanged.

REGISTER TRANSFER DESCRIPTION:

(RA, RA+1) : DO;

(CS) <-- 0010 if (RA, RA+1) = DO;

(CS) <-- 0001 if (RA, RA+1) < DO;

(CS) <-- 0100 if (RA, RA+1) > DO;

REGISTERS AFFECTED: CS

5.94 Floating point compare.

ADDR	MODE	MNEMONIC	FORMAT/OPCODE
R	FCR	RA, RB	844 ----- F9 RA RB -----
B	FCB	BR, DSPL	4228 ----- 3 3 BR' DSPL ----- $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
BX	FCBX	BR, RX	42244 ----- 4 0 BR' D RX ----- $12 \leq BR \leq 15$ $BR' = BR - 12$ $RA = R0$
D DX	FC FC	RA, ADDR RA, ADDR, RX	84416 ----- F8 RA RX ADDR -----

DESCRIPTION: The floating point number in registers RA and RA + 1 is compared to the floating point Derived Operand, DO. Then, the Condition Status, CS, 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 and RA + 1 are unchanged.

Note: This instruction does not cause an overflow to occur.

REGISTER TRANSFER DESCRIPTION:

(RA, RA+1) : DO;

(CS) <-- 0010 if (RA, RA+1) = 0;
 (CS) <-- 0001 if (RA, RA+1) < 0;
 (CS) <-- 0100 if (RA, RA+1) > 0;

REGISTERS AFFECTED: CS

MIL-STD-1750A (USAF)
2 July 1980

5.95 Extended precision floating point compare.

<u>ACDR</u>	<u>MODE</u>	<u>MNEMONIC</u>	<u>FORMAT/OPCODE</u>			
			8	4	4	
R		EFCR RA, RB	FB	RA	RB	
D		EFC RA, ADDR	8	4	4	16
DX		EFC RA, ADDR, RX	FA	RA	RX	ADDR

DESCRIPTION: The extended precision floating Derived Operand, DO, is compared to the contents of registers RA, RA + 1, and RA + 2 where RA contains the most significant 16-bits of the extended precision floating point word. The condition status, CS, is set based on whether the contents of RA, RA + 1, and RA + 2 are less than, equal to or greater than the DO. The contents of RA, RA + 1, and RA + 2 are unchanged.

Note: This instruction does not cause overflow to occur.

REGISTER TRANSFER DESCRIPTION:

(RA, RA+1, RA+2) : DO;

(CS) <-- 0010 if (RA, RA+1, RA+2) = DO;

(CS) <-- 0001 if (RA, RA+1, RA+2) < DO;

(CS) <-- 0100 if (RA, RA+1, RA+2) > DO;

REGISTERS AFFECTED: CS

5.96 No operation.

ADDR MODE MNEMONIC

FORMAT/OPCODE

S NOP

8	4	4
FF	0	0

DESCRIPTION: No operation is performed.

REGISTER TRANSFER DESCRIPTION: None

REGISTERS AFFECTED: None

MIL-STD-1750A (USAF)
2 July 1980

5.97 Break point.

ADDR MODE MNEMONIC

FORMAT/OPCODE

S BPT

8	4	4

	FF	F F

DESCRIPTION: This instruction is typically used for halting the processor during maintenance and diagnostic procedures when the maintenance console is connected to the system. If the console is not connected, this instruction is treated as a NOP (see page 137). Restarting the processor after a BPT can only be done by: the maintenance console or the power on sequence.

REGISTER TRANSFER DESCRIPTION: None

REGISTERS AFFECTED: None

MIL-STD-1750A (USAF)
2 July 1980

Custodian:
Air Force - 11

Reviewing activity:
Air Force - 02

Preparing activity:
Air Force - 11

Project IPSC-F142

INDEX

Name Page

A	99
AB	89
ABS	92
ABX	89
AIM	89
AISP	89
AND	123
ANDB	123
ANDM	123
ANDR	123
ANDX	123
AR	89
BEX	62
BEZ	60
BGE	66
BGT	64
BLE	63
BLT	61
BNZ	66
BPT	138
BR	59
C	132
CB	132
CBL	133
CBX	132
CI	31
CIM	132

CISM	132
CISP	132
CLC	30
CLTR	29
CO	30
CR	132
D	117
DA	94
DABS	93
DAR	94
DB	117
DBX	117
DC	134
OCR	134
DD	118
DDR	118
DECM	101
DIM	117
DISM	116
DISP	116
DL	72
DLB	72
DLBX	72
DLI	72
DLR	72
DM	111
DMAD	30
DMAE	30
DMR	111
DNEG	103
DR	117
DS	104
DSAR	53

MIL-STD-1750A (USAF)
2 July 1980

DSBL	29
DSCR	54
DSLC	48
DSLL	45
DSLR	52
DSR	104
DSRA	47
DSRL	46
DST	81
DSTB	81
DSTI	81
DSTX	81
DSUR	30
DV	116
DVIM	116
DVR	116

EFA	97
EFAR	97
EFC	136
EFCR	136
EPD	121
EFDR	121
EFIX	128
EFL	74
EF.LT	129
EFM	114
EFMR	114
EFS	107
FFSR	107
EFST	84
FNBI	29
ESUR	30

MIL-STD-1750A (USAF)
2 July 1980

FA 95
FAB 95
FABS 98
FABX 95
FAR 95
FC 135
FCB 135
FCBX 135
FCF 135
FD 119
FDB 119
FDBX 119
FDR 119
FID 126
FID 127
FM 112
FMB 112
FMBX 112
FMR 112
FNEG 108
FS 105
FSB 106
FSBX 106
FSP 105

6 30

7 30

8 30

9 30

10 30

11 30

MIL-STD-1750A (USAF)
2 July 1980

L	70
LB	70
LBX	70
LI	70
LIM	70
LISN	70
LISP	70
LIB	76
LIBI	76
LM	73
LMP	31
LR	70
LST	67
LSTI	67
LUB	75
LUBI	75

M	110
MB	110
MBX	110
MIM	110
MISN	109
MISP	109
MOV	90
MPEN	30
MR	110
MS	109
MSIM	109
MSR	109

N	125
NIG	102
NIM	125

AD-A100 577

AERONAUTICAL SYSTEMS
AFSC STANDARDIZATION
NOV 80 E C GANGH, S E SMITH
ASD-TR-80-5050-VOL-2

DIV WRIGHT-PATTERSON AFB OH
CONFERENCE, 1553, 1589, 1750, 1760, ADA, N-ETC(U)

F/G 1/3

UNCLASSIFIED

NL

5 of 5
AD-A100 577

END
DATE
FILMED
7-8-81
DTIC

MIL-STD-1750A (USAF)
2 July 1980

NOP	137
NR	125
OD	30
OR	122
ORB	122
ORBX	122
ORIM	122
ORR	122
OTA	30
OTB	31
PI	29
PO	29
POPM	77
PSHM	87
RB	35
RBI	35
RBR	35
RCFR	30
RCS	31
RDI	31
RDOR	31
RIC1	31
RIC2	31
RIPR	32
RMFS	31
RMK	30
RMP	32
RNS	30
ROPR	32
RPI	29
RPIR	30

MIL-STD-1750A (USAF)
2 July 1980

RSW	30
RVBR	39
S	39
SAR	60
SB	14
SBB	69
SBBX	19
SBI	34
SBR	34
SCR	51
SLM	99
SLSP	99
SJS	68
SLBI	86
SLC	44
SLI	11
SLR	49
SMK	29
SOJ	58
SPC	19
SR	79
SRA	13
SP	12
SM	32
S	78
SB	18
SBX	14
SD	79
SDI	79
SDI	18
SDR	10
SM	23
SDR	10

MIL-STD-1750A (USAF)
2 July 1980

STZ 79
STZI 79
SUBI 85
SVBR 38

TAH 30
TAS 30
TB 36
TBH 31
TBI 36
TBR 16
TBS 31
TPIO 31
TSB 37
TVBR 40

URS 69

VIO 33

WIPR 31
WOPR 31
WSW 29

XBR 130
XIO 29
XOR 124
XORM 124
XORR 124
XWR 131

MIL-STD-1760 (Draft)
May 1980

MILITARY STANDARD

STANDARD STORE INTERFACE,

AIRCRAFT/STORES ELECTRICAL

INTERFACE DEFINITION

MIL-STD-1760 (Draft)

DEPARTMENT OF THE AIR FORCE

WASHINGTON D.C. 20330

MIL-STD-1760 (Draft) STANDARD STORES INTERFACE,
AIRCRAFT/STORES ELECTRICAL INTERFACE DEFINITION

1. This Military Standard has been approved for use by all Departments and Agencies of the Department of Defense.
2. Recommended corrections, additions, or deletions should be addressed to U.S. Air Force Armament Laboratory, Attention: AFATL/DLJA, Eglin Air Force Base, Florida 32542

FOREWORD

1. Prior to this standard, an aircraft and the stores which it carried were typically developed independently of each other or were developed exclusively for each other. The usual results were new aircraft/store electrical interface requirements and the general proliferation of overall store interface requirements. The lack of standards within DoD for an aircraft/store electrical interface led to low levels of interoperability and costly aircraft modifications to achieve required store utilization flexibility. The trend in store technology toward more complex store functions which require increasing amounts of avionics data from aircraft systems was predicted to produce insurmountable aircraft/store interface problems.
2. This standard contains a solution to aircraft/store interface implementation proliferation by specifying one standard electrical interface for all future aircraft and stores. The interface contained herein is based on recognized trends in stores management systems which predict the use of serial digital transmission for control, monitor, and release signals to the store station. Application of this standard to new aircraft and stores will serve to reduce and stabilize the number and variety of signals required at the aircraft/store interface, minimize the impact of new stores on future stores management systems, and increase store interoperability within the services and NATO.
3. After initial implementation of the standard, there will be a period when both standard and non-standard aircraft and stores will coexist in the inventory. Therefore, in order to be compatible, some development aircraft and stores may be built with both the standard and non-standard interfaces.
4. The format of this standard is aligned to aircraft/store interface control documents used by industry to describe aircraft/store interface requirements to DoD.
5. Unless otherwise specified, this standard conforms to the Nuclear System 2, Digital Interface Requirements document (which is now under development with the Department of Energy and the National Laboratories) and appropriate DoD agencies.

TABLE OF CONTENTS

Paragraph		Page
1.	INTRODUCTION	1
2.	REFERENCED DOCUMENTS	1
3.	DEFINITIONS	3
4.	GENERAL STATEMENTS OF REQUIREMENTS	4
4.1	Introduction	4
4.2	Standard Interface System	4
4.2.1	Digital Multiplex Data Buses	4
4.2.2	Aircraft Power	4
4.2.3	High Bandwidth Signals	4
4.2.4	Store Address Lines	4
4.3	Store Implications	6
4.4	Stores Management System	6
5.	DETAILED STATEMENTS OF REQUIREMENTS	6
5.1	Introduction	6
5.2	Electrical Interface Definition	8
5.2.1	Digital Multiplex Data Bus	11
5.2.1.1	Bus Operation	11
5.2.1.1.1	Subaddress/Mode	11
5.2.1.1.2	Mode Control	11
5.2.1.1.3	Logical Interface Code	11
5.2.1.1.4	Information Transfer Modes	11
5.2.1.2	Store On Data Bus	13
5.2.1.3	Aircraft On Data Bus	13
5.2.1.4	Electrical Interface Requirement	13
5.2.1.5	Data Bus Requirement	13
5.2.1.6	Electromagnetic Compatibility	13
5.2.2	Fiber Optics Data Bus	13
5.2.3	High Bandwidth Signals	13
5.2.3.1	Video	13
5.2.3.2	Time Correlation Pulse (TCP)	14
5.2.3.3	Radio Frequency	14
5.2.4	Digital Address Lines	14
5.2.5	Aircraft Power	14
5.2.5.1	DC Power	14
5.2.5.1.1	Voltage	14
5.2.5.1.2	Over-voltage	14
5.2.5.1.3	Current	15
5.2.5.1.3.1	Power 1, Power 2, and Emergency Jettison Lines	15
5.2.5.1.3.2	Power 3 Lines	15
5.2.5.1.3.3	Initial Store Power	15
5.2.5.1.4	Isolation	15

TABLE OF CONTENTS (Continued)

Paragraph		Page
5.2.5.1.5	Return Lines	15
5.2.5.1.6	Rise Time	15
5.2.5.1.7	AC Power.	15
5.2.5.1.8	Voltage	15
5.2.5.2.2	Over-voltage	15
5.2.5.2.3	Current	15
5.2.5.2.3.1	SSI Connector.	15
5.2.5.2.3.2	Auxiliary Power Connector.	15
5.2.5.2.4	Isolation.	15
5.3	Physical Interface Definition.	16
5.3.1	Wiring and Cabling	16
5.3.2	Wiring and Shielding Termination and Grounding.	16
5.3.3	Electromagnetic Interference	16
5.3.4	Static Discharge Survival.	16
5.3.5	Environmental Factors.	16
5.3.6	Store Receptacle Connector Orientation and Mounting	16
5.3.6.1	Top Mounting	16
5.3.6.2	Rear Mounting.	16
5.3.6.3	Mounting Alignment and Integrity	16
5.3.6.4	Receptacle Position.	18
5.3.7	Insert Arrangement	18
5.3.8	Exposed Receptacle (separation).	18
5.4	Logical Interface Definition	18

FIGURES

Figure		Page
1.	Aircraft/Store Electrical Interface.	5
2.	Typical Store/Data Bus Electronics Functional Diagram	7
3a.	Electrical Signal Set - SSI Connector.	9
3b.	Electrical Signal Set - Auxiliary Power Connector.	10
4.	Standard Interface Employment Architecture	12
5.	Master Keyway Position	17
6a.	SSI Connector - insert arrangement	19
6b.	Auxiliary Power Connector - insert arrangement.	20

TABLE

Table		Page
1	Electrical Signals Sets	21
	SSI Connector	22
	Auxiliary Power Connector	24

APPENDIX A

Paragraph		Page
6.	ELECTRICAL CONNECTOR REQUIREMENTS	A-1
6.1	Introduction	A-1
6.2	Definitions	A-1
6.2.1	Electrical Connector Terminology	A-1
6.2.2	Requirements	A-1
6.3	General Requirements	A-1
6.3.1	Design and Construction	A-1
6.3.2	Types	A-1
6.3.3	Finish	A-1
6.3.4	Interchangeability and Intermateability	A-2
6.3.5	Pin Protection	A-2
6.3.6	Coupling	A-2
6.3.6.1	Ease of Coupling	A-2
6.3.7	Engagement and Locking	A-2
6.3.8	Polarization of Connector Shells	A-2
6.3.8.1	Pin to Pin Mating Prevention	A-2
6.3.9	Electrical Continuity	A-2
6.3.10	EMI Grounding Spring Fingers	A-2
6.3.11	Lanyard	A-3
6.3.11.1	Lanyard Retention	A-3
6.3.11.2	Lanyard Release Force	A-3
6.3.12	Shielding Braid Termination	A-3
6.3.13	Water Sealing	A-3
6.3.14	Electrical Contacts	A-3
6.3.14.1	Contact Sizes	A-3
6.3.14.2	Insert Arrangement	A-3
6.3.14.3	Connector Shell Size	A-3
6.3.15	EMI Effectiveness	A-3
6.3.15.1	EMP Susceptibility	A-4
6.3.16	Receptacle Mounting	A-4
6.3.16.1	Store Mounting	A-4

APPENDIX A (Continued)

Paragraph		Page
6.3.16.2	Aircraft Mounting.	A-4
6.4	Detailed Requirements.	A-4
6.4.1	Materials.	A-4
6.4.1.1	Metals	A-4
6.4.1.2	Dissimilar Metals and Compatible Couples.	A-4
6.4.1.3	Hydrolytic Stability	A-5
6.4.2	Components	A-5
6.4.2.1	Class F.	A-5
6.4.2.2	Fungus Resistant	A-5
6.4.2.3	Nonmagnetic Materials.	A-5
6.4.3	Insert Design.	A-5
6.4.3.1	Environment Resisting Classes.	A-5
6.4.4	Mating Seal.	A-5
6.4.5	Shell.	A-6
6.4.5.1	Spring Fingers	A-6
6.4.5.2	Jam-Nut Mounting Receptacles	A-6
6.4.6	Lubrication.	A-6
6.4.7	Plating.	A-6
6.4.7.1	Contacts (Crimp)	A-6
6.4.7.2	Shell and Accessory Hardware	A-6
6.4.8	Contact Engagement and Separation Force.	A-7
6.4.9	Thermal Shock.	A-7
6.4.10	Coupling Torque.	A-7
6.4.11	Durability	A-7
6.4.12	Altitude Immersion	A-8
6.4.13	Insulation Resistance.	A-8
6.4.13.1	Insulation Resistance at Ambient Temperature.	A-8
6.4.13.2	Insulation Resistance at Elevated Temperature.	A-8
6.4.14	Dielectric Withstanding Voltage.	A-8
6.4.15	Insert Retention	A-8
6.4.16	Salt Spray	A-8
6.4.17	Electrical Engagement.	A-8
6.4.18	External Bending Moment.	A-8
6.4.19	Contact Retention.	A-8
6.4.20	Altitude - Low Temperature	A-8
6.4.21	Vibration.	A-9
6.4.22	Shock.	A-9
6.4.23	Shell-to-Shell Conductivity.	A-9
6.4.24	Humidity	A-9
6.4.25	Shell Spring Finger Forces	A-9
6.4.26	Ozone Exposure	A-9
6.4.27	Fluid Immersion.	A-9

APPENDIX A (Continued)

Paragraph	Page
6.4.28	Retention System Fluid Immersion A-9
6.4.29	Pin Contact Stability A-10
6.4.30	Contact Walkout. A-10
6.4.31	Power Contacts A-10
6.4.31.1	Temperature Life with Contact Loading. A-10
6.4.31.2	Temperature Life A-10
6.4.32	Electrolytic Erosion A-10
6.4.33	Firewall. A-10
6.4.34	Coaxial and Fiber Optics Contacts. A-10
6.4.35	Marking A-11
6.4.35.1	Contact Location Identification A-11
6.4.36	Workmanship A-11

FIGURES

Figure	Page
1.	Receptacle, Wall Mount (Crimp) (Pin Insert) A-12
2.	Receptacle, Box (Crimp) (Socket Insert). A-13
3.	Plug-Straight-Crimp, Pin & Socket. A-14
4.	Receptacle - Plug, Insert Detail (Style S&P). A-15

TABLES

Table	Page
I.	Deleted
II.	EMI Shielding Effectiveness. A-4
III.	Contact Engagement and Separation Forces A-7
IV.	Coupling Torque. A-7
V.	Shell Spring Finger Forces A-9
VI.	Pin Contact Stability. A-10

1. INTRODUCTION

1.1 Scope. This standard establishes the definition for development, application, and control of the electrical interface between stores and their carrying aircraft. It encompasses all aircraft-to-store and store-to-aircraft signal functions, electrical characteristics, and electrical connector components of the interface. It implies, but does not quantify, aircraft/store electrical systems to meet the interface defined herein. It is intended that all aircraft stores which require any electrical interface shall use the same functional interface and the same physical connectors. This document does not include the design of the physical connectors, but has design guides for them. Stores which require a limited subset of the functions will contain the necessary electronics to accept those portions of the functional interface signals required without aircraft hardware modifications. This standard does not imply that all aircraft can carry all stores on all stations. Factors such as physical and mechanical limitations, and electrical power must be considered.

1.2 Purpose. The purpose of this interface standard is threefold: (1) Define specific electrical and optical characteristics of power and data signals to be provided at the interface, (2) Provide guidelines for interface connectors, (3) Define the logical portion of the interface which includes the interface message traffic and information coding formats (TBD).

1.3 Classification. This standard covers all aircraft stores that require an electrical interface. This coverage encompasses both expendable and non-expendable stores as defined in NATO AAP-6. Non-expendable stores include suspension equipment as defined in NATO AAP-6.

1.4 Effectivity Data. Upon date of implementation, this standard is effective with stores and aircraft in concept development stages and all future aircraft and all store development within the classification data provided above.

2. REFERENCED DOCUMENTS

2.1 Issues of Documents. The following documents of the issue in effect on date of invitation for bid or request for proposal form a part of this standard to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this standard, the contents of this standard shall be considered a superseding requirement.

SPECIFICATIONS

MILITARY:	MIL-E-6051	-	Electromagnetic Comp Reqmts System
	MIL-W-5088	-	Wiring, Aerospace Vehicle
	MIL-A-8591	-	Airborne Stores, Associated Suspension Lugs, and Aircraft Store Interface (Carriage Phase)

- MIL-C-38999 - Connectors, Electrical.
- MIL-C-27599 - Connector, Electrical, Minature, Quick Disconnect (For Weapons Systems) Established Reliability.

STANDARDS

- MILITARY: MIL-STD-461 - Electromagnetic Interference Characteristics, Requirements for Equipment.
- MIL-STD-462 - Electromagnetic Interference Characteristics, Measurement of.
- MIL-STD-704 - Aircraft Electric Power Characteristics.
- MIL-STD-1553 - Aircraft Internal Time Division Command/Response Multiplex Data Bus.

HANDBOOKS

- MILITARY: AFSC DH 1-4 - Electromagnetic Compatibility.

2.2 Other Publications. The following documents form a part of this standard to the extent specified herein. Unless otherwise indicated, the issue in effect on date of invitation for bid or request for proposal shall apply. In the event of conflict between the documents referenced herein and the contents of this standard, the contents of this standard shall be considered a superseding requirement.

- EIA STD RS-330 - Electrical Performance Standards for Closed Circuit Television Camera 525/60 Interlaced 2:1.
- STANAG 3558 - NATO Standardization Agreement. Location of the Electrical Control Connector for Airborne Armament Stores.
- STANAG 3838AA (DRAFT) - NATO Standardization Agreement. Aircraft Internal Time Division Command/Response Multiplex Data Bus.
- EIA STD RS-343A - Electrical Performance Standard for High Resolution Monochrome Closed Circuit Television Camera.
- SYSTEM 2 - Nuclear Digital Interface Requirements.
- NATO AAP-6 - NATO Glossary of Terms and Definitions for Military Use.

3. DEFINITIONS

3.1 Address. A store location identification, as represented by a five-digit binary number.

3.2 Aircraft. For the purpose of the standard, the term aircraft shall be that which serves the function of commanding, scanning, and monitoring bus traffic, and which provides electrical power.

3.3 Bus Interface Electronics. The electronic module necessary to interface the data bus with the subsystem and the subsystem with the data bus.

3.4 EIA. Electronic Industries Association.

3.5 EMI. Electromagnetic Interference.

3.6 EMC. Electromagnetic Compatibility.

3.7 EMP. Electromagnetic Pulse.

3.8 GPS. Global Positioning System.

3.9 Message. A message is a time sequential transmission of words on the data bus. A message transfer is complete when the command word, data word(s) and the status word(s) have been transmitted. There are three types of messages: (1) Aircraft-to-store, (2) Store-to-aircraft, and (3) Store-to-store.

3.10 Electrical Interface. The connectors through which the data signals and power flow between the aircraft and store.

3.11 Bit. Contraction of binary digit; may be either 0 (zero) or 1 (one). It is equal to one binary decision or the designation of one of two possible values or states of anything used to store or convey information.

3.12 Bit Rate. The number of bits transmitted per second.

3.13 Half Duplex. Data transfer in either direction over a single line but not in both directions on that line simultaneously.

3.14 Command/Response Mode. Operation of data link in which the store will respond only when commanded by the aircraft.

3.15 Asynchronous - Operation. An independent clock source in each store which is utilized for the transmission of messages. The received messages shall be decoded using clock information derived from the received signal.

3.16 Aircraft/Store Interface. The electrical (connector) interface between the aircraft electrical system and the store electrical umbilical cable.

3.17 Byte. Each half of a 16-bit word (8bits) is a byte.

3.18 Store. Same as aircraft store in NATO AAP-6, repeated here for convenience. Any device intended for internal or external carriage and mounting on aircraft suspension and release equipment, whether or not the item is intended to be separated in flight from the aircraft. Aircraft stores are classified in two categories as follows:

a. Expendable store - an aircraft store normally separated from the aircraft in flight such as a missile, rocket, bomb, nuclear weapon, torpedo, pyrotechnic device, sonobuoy, signal, underwater sound device, and other similar items.

b. Non-expendable store - an aircraft store which is not normally separated from the aircraft in flight such as a tank (fuel and spray), line-source disseminator, pods (refueling, thrust augmentation, gun, electronic-countermeasure, data link, etc.), target, cargo drop container, drone and other similar items.

NOTE: For the purposes of this standard, non-expendable stores include suspension equipment (racks, adapters, missile launchers, etc). It excludes thrust augmentation devices.

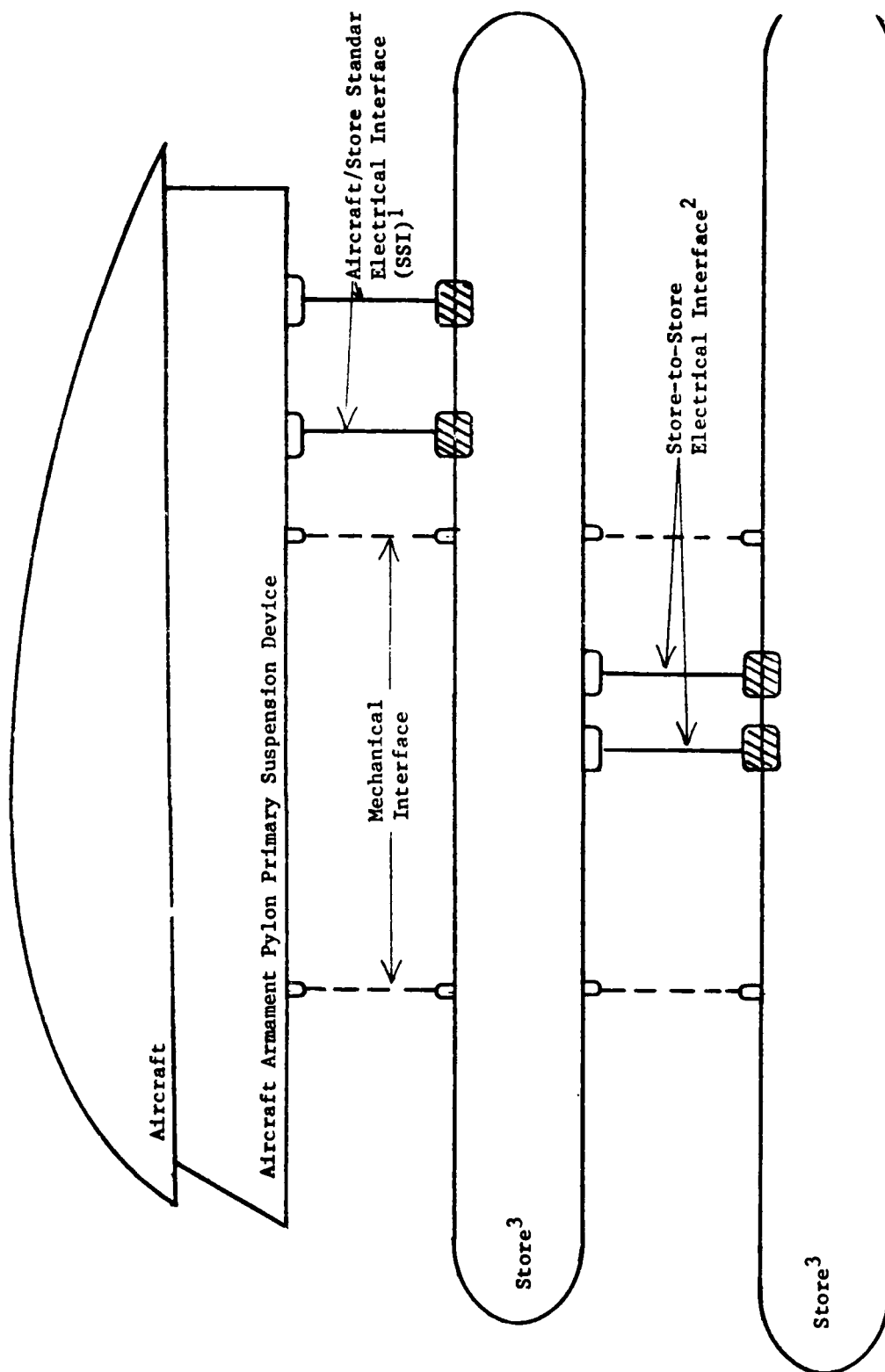
3.19 TCP. Time Correlation Pulse.

3.20 TBD. To Be Determined.

4. GENERAL STATEMENTS OF REQUIREMENTS

4.1 Introduction. This standard defines an aircraft-to-store electrical interface in terms of its physical, electrical, and logical elements. The aircraft/store interfacing is accomplished through a maximum of two electrical connectors per store and is defined at the store receptacle disconnect and at the aircraft skin (see Figure 1). This standard addresses only the aircraft-to-store interface, however, it is intended that this standard interfacing concept will also be applied to electrical interfaces between stores (e.g., launcher-to-missile) and will be addressed in further versions of the MIL-STD. In this standard, the interface is categorized and defined in three generic parts: (1) Electrical (voltage, current, resistance, isolation, risetime, etc), (2) Physical (connector characteristics, pin size, mounting, etc), and (3) Logical (information content, protocols, timing, etc) - TBD.

4.2 Standard Interface System. The standard interface system consists of a composite set of electrical/optical circuits, and specific interface connectors. The signal set is comprised of redundant digital multiplex data buses, coaxial contacts for high bandwidth signals, aircraft AC and DC power, digital address lines, and a limited number of dedicated hardwired discretes. Provisions are included for digital multiplex data transfer over twisted shielded pairs and optionally through fiber optic cable.



1. Includes SSI/Auxiliary Power Connectors.
2. This electrical interface between stores is presently not defined but will be included in future versions of this standard.
3. Store can be expendable (i.e. missile, bomb, etc) or non-expendable (rack, pod, etc).

Figure 1. Aircraft/Store Electrical Interface

Interface signals (electrical) are specified in 5.2. Interface connector (physical) requirements and environmental conditions are specified in 5.3 and Appendix A. Information interface (logical) will be specified in 5.4. (TBD)

4.2.1 Digital Multiplex Data Buses. The majority of all command, control and status data transfer between the store and the aircraft is performed through redundant digital multiplex data buses as specified in 5.2.1 and described by MIL-STD-1553. Store connection to the aircraft data bus is accomplished using stubbing methods also prescribed in MIL-STD-1553. Data buses are operated by the aircraft in an active standby redundant mode. The aircraft commands the store to perform functions and transmit store status using specially formatted digital messages. The interface defined herein is intended to provide sufficient capacity and performance to satisfy the requirements of all future store types.

4.2.2 Aircraft Power. The aircraft supplies DC and AC power to the store interface in accordance with MIL-STD-704. Interface power requirements and exceptions to MIL-STD-704 are specified in 5.2.5.

4.2.3 High Bandwidth Signals. Coaxial and fiber optic contacts are provided in the standard interface connector to accommodate high bandwidth (RF, Video, Audio) interface signals. User guidelines for these interface circuits are specified in 5.2.3.

4.2.4 Store Address Lines. Dedicated hardwire circuits that provide a unique store address. Detail characteristics of these circuits are specified in 5.2.4.

4.3 Store Implications. The interface prescribed by this standard requires that each store contain electronic circuitry necessary to interface the store directly onto the aircraft's digital data bus. This store electronics must be capable of receiving and transmitting digital multiplex data as defined in MIL-STD-1553 and be capable of converting digital data into required store commands. A functional diagram of representative store circuitry is contained in Figure 2.

4.4 Stores Management System Architecture. The aircraft must be capable of transmitting and receiving digital multiplex data to a store controller as defined in MIL-STD-1553. This standard does not prescribe, or recommend a particular stores management system architecture. The standard is designed to be applicable to a wide variety of stores management schemes. Viable schemes found to be compatible with this standard should be reported to the U.S. Air Force Research Laboratory, Eglin Air Force Base, Florida.

5. DETAILED INTERFACE REQUIREMENTS

5.1 Introduction. This section prescribes detail electrical and physical requirements for the standard aircraft/store interface. Signal functions, electrical characteristics and connector pin assignments are specified, as

are specified and operating requirements for the standard interface.

Electrical Interface Definition. All aircraft electrical interface requirements with allotted stores shall be accomplished through the electrical signal sets contained in Figures 3a and 3b, and defined in Table 1. The particular pin assignment and arrangement for the SSI and Auxiliary Power connectors are provided for identification and traceability purposes only. Definitive signal and connector characteristics will be provided in further versions of this MIL-STD. Also, spare pins, which are required for future growth, are not shown and will be addressed later. Signals supplied by these two connectors are arranged in descending order, starting with high bandwidth signals and ending with structural ground. Five signal types are contained in the interface: (1) High bandwidth signals, (2) Serial digital data buses, (3) Store address lines, (4) Dedicated discrete lines, and (5) High and low aircraft power. Each interface line is defined in Table 1 with one line per page. The pages are arranged alphabetically by pin letter(s) for the SSI connector, and by pin number(s) for the Auxiliary Power connector for quick reference. Each signal description follows the standard format defined below:

PIN:	Each interface connector pin in the SSI connector (Figure 3a) is assigned an alphabetic nomenclature beginning with upper case and continuing through the lower case letters. The letters I, O, I, and l are not used to avoid possible confusion. In the Auxiliary Power connector (Figure 3b) each pin is assigned a number.
TITLE:	Specifies signal name.
FUNCTION:	Describes the function performed by the signal in the store or carrier aircraft.
SOURCE:	Indicates the source of the signal (signal transmitter).
DESTINATION:	Indicates the destination of the signal (signal receiver).
RETURN:	Cross-references the return path for the signal being described.
CONTACT:	Describes the contact size. It also describes special wire treatment such as twisted pair or shielding.
CHARACTERISTICS:	Indicates type of signal and specifies signal characteristics such as range, scale factor and polarity or phasing. Except as noted, aircraft power signals are specified to meet MIL-STD-704

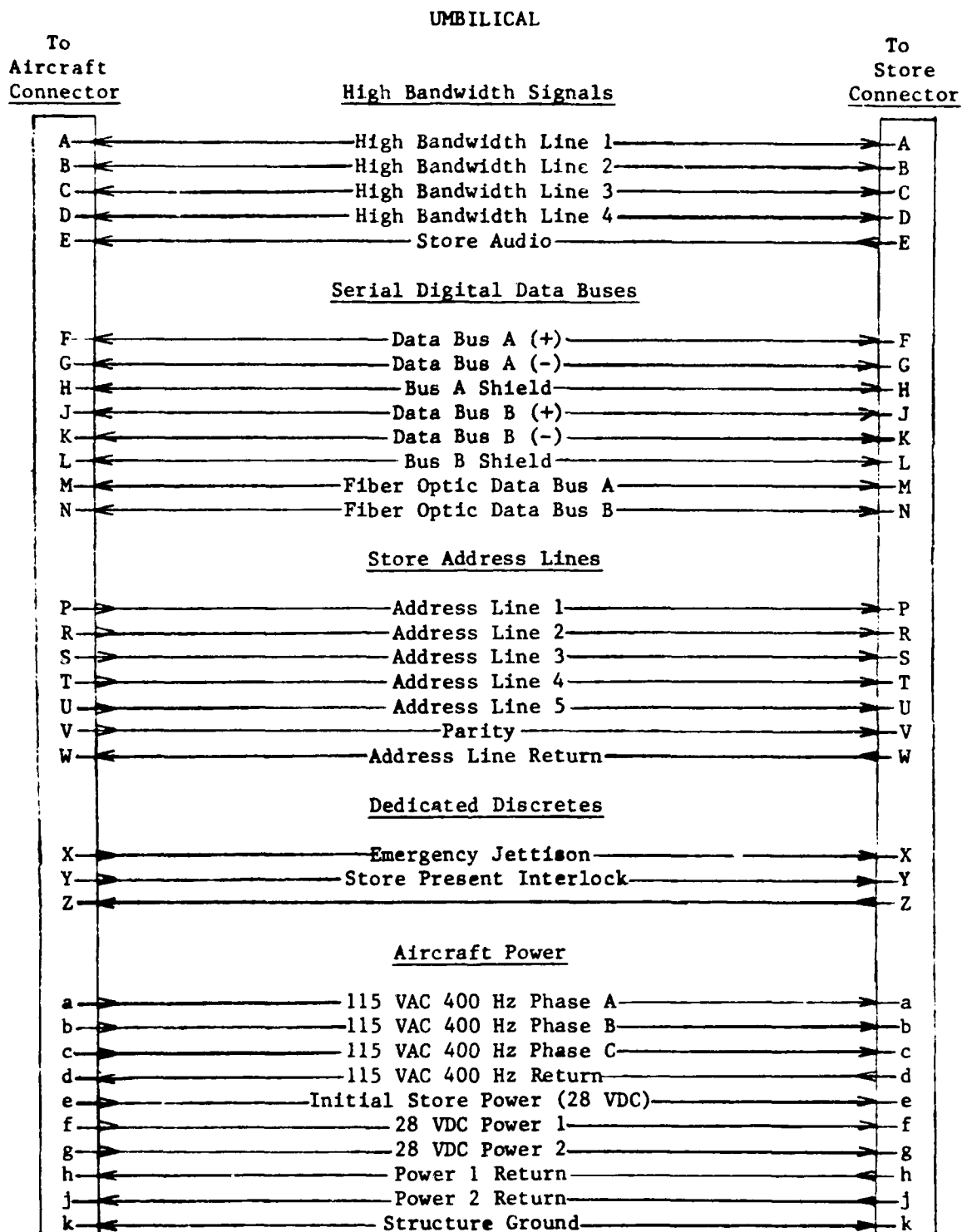


Figure 3a - Electrical Signal-Set SSI Connector

UMBILICAL

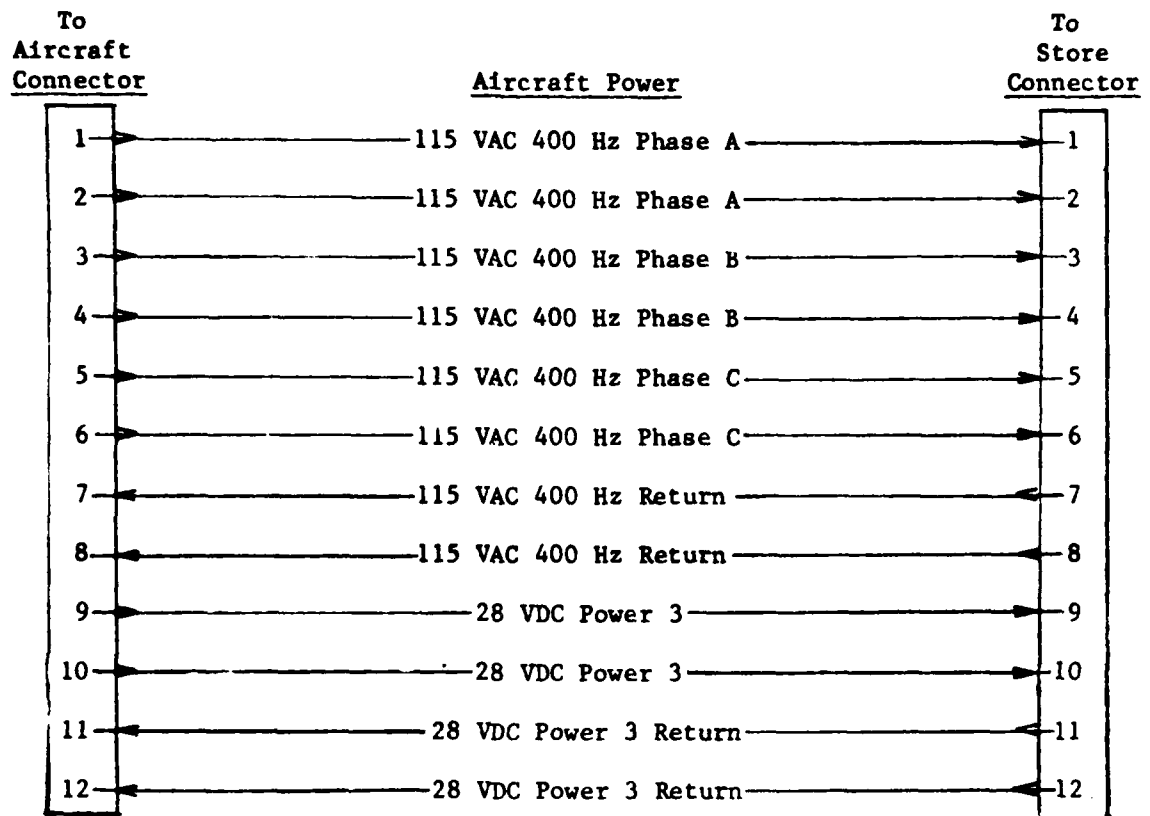


Figure 3b - Electrical Signal Set-Auxiliary Power Connector

standards. Discrete signals are always binary and both voltage and logic states are described.

LOAD: Specifies the electrical current load requirement. Load or source impedances in the weapon when specified are resistive unless otherwise stated.

REMARKS: Provides additional comments, safety isolation requirements, references and/or other pertinent information.

5.2.1 Digital Multiplex Data Bus. The functions of pins F, G, H and J, K, L are to provide stub connections to two dual redundant bidirectional serial digital multiplex buses which shall be used to transfer data between the aircraft and the store. The store shall contain bus interface electronics configured to interface two independent data buses to one set of store subsystem signals. Serial data words and synchronization are combined by means of Manchester biphasic coding into one signal transmitted over a shielded, twisted pair of conductors. The following paragraphs specify bus interface requirements that shall be satisfied by both the aircraft and the store. Requirements pertain to both digital buses.

5.2.1.1 Data Bus Operation. The information flow on the data buses shall be comprised of messages that are formed by three types of words (command, data, status) as defined in para 4.3 of MIL-STD-1553. The aircraft/store multiplex data buses in their most elemental configuration are shown in Figure 4a; more complex configurations, as illustrated in Figures 4b and 4c, will be defined in further versions of this standard. The data buses shall function asynchronously in a command/response mode with transmission occurring in a half-duplex manner. Sole control of message transmission on the buses shall reside with the aircraft. Data format, content, protocol, and timing for these messages will be completely specified in further versions of this interface standard.

5.2.1.1.1 Subaddress/Mode. The subaddress/mode field in the command word illustrated in Figure 3 and paragraph 4.3.3.5.1 of MIL-STD-1553 is hereby defined for application to aircraft/store interfaces.

5.2.1.1.1.1 Mode Control. The five bits following the transmit/receive bit shall indicate mode control when the value is 00000 or 11111 as indicated in MIL-STD-1553.

5.2.1.1.1.1.1 Logical Interface Code. When the five bits following the transmit/receive bit are other than 00000 and 11111, they shall be used as Logical Interface Codes in accordance with paragraph 5.4 TBP which will define the logical portion of the standard interface.

5.2.1.1.1.1.1.1 Information Transfer Mode. The data bus may employ three possible modes of information transfer: (1) aircraft-to-store transfer, (2) store-to-aircraft transfer, and (3) store-to-store transfer. These modes will operate as described in MIL-STD-1553, paragraph 4.3.3.6.

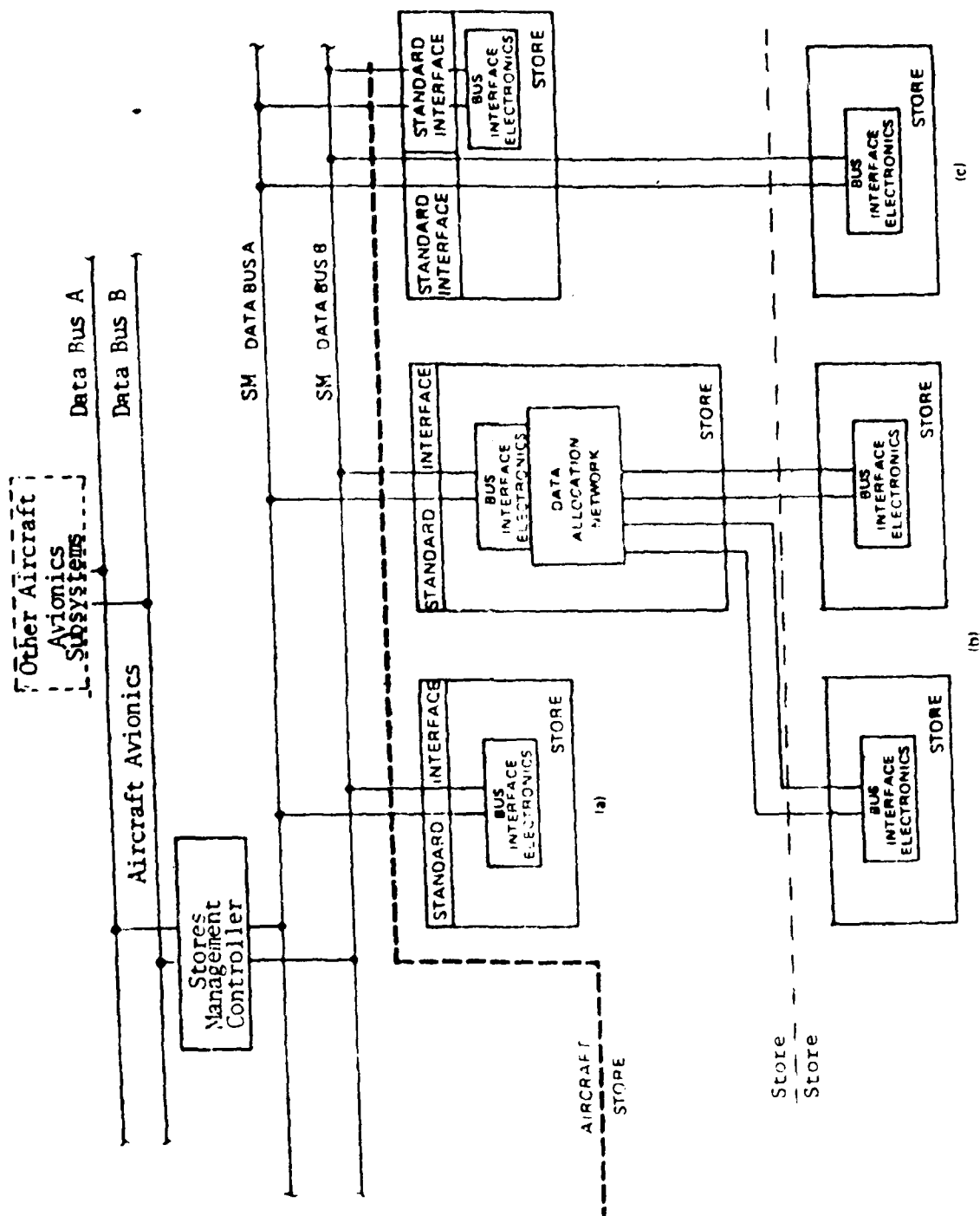


FIGURE 4. Standard interface architecture.

5.2.1.2 Store on Data Bus. The store shall meet the requirements for a remote terminal as defined in MIL-STD-1553 and shall respond to the aircraft stores management data bus as a remote terminal.

5.2.1.3 Aircraft on Data Bus. The aircraft shall be responsible for sending data bus commands, participating in data transfer, receiving status response and monitoring system status as defined for the bus controller in MIL-STD-1553.

5.2.1.4 Electrical Interface Requirements. The store shall meet the electrical requirements of a remote terminal interface with a transformer coupled stub as defined in paragraph 4.5.1 of MIL-STD-1553. The aircraft shall meet the electrical interface requirements of a transformer coupled stub as defined in paragraph 4.5.2 of MIL-STD-1553.

5.2.1.5 Data Bus Requirements. The requirements for redundant data buses as specified in paragraph 4.6 of MIL-STD-1553 shall apply.

5.2.1.6 Electromagnetic Compatibility. The generation of and susceptibility of electromagnetic interference shall be controlled in the store and aircraft. The design shall meet MIL-STD-461, Notice 3 and shall be tested in accordance with MIL-STD-462, Notice 2. The specific susceptibility requirements of MIL-STD-461, Notice 3 listed below shall be included as a minimum.

Electric Field - RS03

Magnetic Field - RS02

5.2.2 Fiber Optic Data Bus. The standard interface contains a provision for two fiber optic contacts to accommodate technology advances that are expected to produce DoD approved and sanctioned fiber optic digital data buses for aircraft and stores. Standards have not been established for transmission of serial digital multiplex data over fiber optic data buses. As standards evolve, they will incorporate in part, or in total, into this standard, and will be accommodated in either the physical connector suggested herein, or in a separate physical connector.

5.2.3 High Bandwidth Signals. The SSI connector shall contain four coaxial contacts for bidirectional transfer of high bandwidth signals between the aircraft and stores, and a fifth contact for transmission of store audio data. The aircraft (SMS) shall assign, monitor, control, and route these signals to their proper destination. Typical high bandwidth line application includes: video, very high speed digital and analog, and Radio Frequency (RF) signals. When video, Time Correlation Pulse, or RF signals are required, they shall be allocated as follows:

5.2.3.1 Video. High Bandwidth Line 1 (and continuing in order of priority through Line 4, if needed) shall be used for the bidirectional transfer of video type signals between the aircraft and stores. The electrical charac-

teristics of the video shall be in accordance with EIA Standard RS-330, with the following exceptions:

1. Sync pulse amplitude shall be $.8 \pm .08$ volts.
2. Video sensor load impedance shall be 93 ohms, $\pm 10\%$.
3. Signal can be bidirectional.
4. Video shall be 525 lines, or 875 lines, in accordance with EIA Standard RS-343A.

5.2.3.2 Time Correlation Pulse. TCP signal when required shall be assigned to High Bandwidth Line 3. The function of this circuit is to transfer differential digital target signals from the store. Specific characteristics of the TCP signal are TBD.

5.2.3.3 Radio Frequency. RF signals, when required, shall be assigned to High Bandwidth Line 4. Typically, this circuit transfers Global Positioning System (GPS) RF data from the aircraft to the store's navigational subsystem.

5.2.4 Store Address Lines. The aircraft shall supply address line outputs (5 interface address lines, 1 parity line, and 1 address line return) to each store interface. A logic "one" shall be an open circuit ($\geq 100K$ ohms DC) referenced to the address line return. A logic "zero" shall be ≤ 10 ohms DC referenced to the address line return. Address assignments shall be made such that each store station has a separate and unique address. Each assigned address plus parity shall contain an odd number of "ones".

5.2.5 Aircraft Power. Application and control of all power to the store shall reside within the aircraft (normally a function of the stores management system). Aircraft power is specified for both the SSI (Figure 3a) and the Auxiliary Power (Figure 3b) connectors. However, the Auxiliary Power connector shall only be used for unique high power applications (i.e., ECM pods). Not all stores will require or have provisions for handling this power, nor will all aircraft stations be able to supply this high power. The maximum total aircraft power required by a store in any operating mode shall not exceed the power requirements specified below. In addition, aircraft power shall be applied to the interface for equal to or greater than 100 milliseconds prior to its intended use by the store. Detailed aircraft store power requirements are specified as follows:

5.2.5.1 DC Power. MIL-STD-704 shall apply with the following exceptions:

5.2.5.1.1 Voltage. The aircraft shall be capable of supplying 28 volts $\pm 1V$ within the current limitations defined in 5.2.5.1.3.

5.2.5.1.2 Over-voltage. The aircraft shall provide interface protection against over-voltages in excess of 48 volts. Over-voltages between 30 and 48 volts shall be limited to 10 milliseconds or less. Over-voltages of less than 30 volts shall be limited to 100 milliseconds.

5.2.5.1.3 Current. The aircraft shall be capable of providing the following current within the voltage limitations defined in 5.2.5.1.1.

5.2.5.1.3.1 Power 1, Power 2 and Emergency Jettison Lines. 0 to 10 amperes (max) per line.

5.2.5.1.3.2 Power 3 Lines. 0 to 23 amperes (max) per line.

5.2.5.1.3.3 Initial Store Power. 0 to 1 ampere (max).

5.2.5.1.4 Isolation. Power 1, Power 2, Power 3, Emergency Jettison, and Initial Store Power lines shall be isolated at the aircraft interface and controlled independently by the aircraft. Output voltage per line shall not exceed 1 volt into a 1 megohm load and 1 milliamperes into a short circuit when that line is in the Power Off condition, and the other power lines are in either the Power On or Power Off condition.

5.2.5.1.5 Return Lines. Power 1, Power 2, and Power 3 shall have separate and isolated returns within the interface, however, Initial Store Power Return and Power 1 Return are common. Also, Emergency Jettison Return and Power 2 Return are common.

5.2.5.1.6 Rise Time. The rise time for all DC power lines shall not exceed 1 millisecond from 1 volt to 23.75 volts under the conditions stated in 5.2.5.1.1 and 5.2.5.1.3.

5.2.5.2 AC Power. MIL-STD-704 shall apply with the following exceptions:

5.2.5.2.1 Voltage. The aircraft shall be capable of supplying 115 volts AC, $\pm$ 5%, 400 Hertz, three-phase, 4 wire, wye connected power within the current limitations defined in 5.2.5.2.3.

5.2.5.2.2 Over-voltage. The aircraft shall provide interface protection against over-voltages in excess of TBD volts. Over-voltages between TBD volts and TBD volts shall be limited to TBD milliseconds or less. Over-voltages of less than TBD volts shall be limited to TBD seconds.

5.2.5.2.3 Current. The aircraft shall be capable of providing the following current within the voltage limitations defined in 5.2.5.1.1.

5.2.5.2.3.1 Phase 1, Phase 2, and Phase 3. 0 to 10 amperes (max) per phase.

5.2.5.2.3.2 Auxiliary Power Connection. 0 to 10 amperes (max) per line, 40 amperes (max) per phase.

5.2.5.2.4 Isolation. Phase 1, Phase 2, and Phase 3 power lines and the SSI and Auxiliary Power Connection shall be isolated at the aircraft interface and controlled independently by the aircraft. Output voltage per phase line shall not exceed 1 volt into a 1 megohm load and 1 milliamperes into a short circuit when that line is in the Power Off condition, and the other phase lines are in either the Power On or Power Off condition.

5.3 Physical Interface Definition. The aircraft/store physical interface consists of the SSI and Auxiliary Power connectors, and the interconnecting umbilical harness. Detailed connector requirements are contained in Appendix A. Specific physical interface characteristics are as follows:

5.3.1 Wiring and Cabling. The selection and installation of wiring and cabling as it pertains to the scope of this interface shall be in accordance with MIL-W-5088.

5.3.2 Wiring and Shielding Termination and Grounding. The termination of wiring and shielding and the fabrication of cable harnesses, as it affects the integrity of electrical signals through this interface, shall be in accordance with AFSC DH 1-4, Chapter 5, Section 5B, Design Note 5B5, unless otherwise specified herein.

5.3.3 Electromagnetic Interference. The cabling and shielding terminating at this interface, as it affects the electrical integrity of the signals, shall meet the requirements of MIL-STD-461, Notice 3, Test Method RS03, 200 volts per meter test level per paragraph 6.19.2. Shielding braid attenuation shall be 65 DB or greater from 14 KHz to 10 GHz.

5.3.4 Static Discharge Survival. Static discharge of 500 pf capacitor at ± 20 kilovolts through a 5-kohm resistor to an interface connector shell shall not damage the interface.

5.3.5 Environmental Factors. Those components and associated wiring of the aircraft/store configuration which affect the integrity of electrical signals through the interface shall meet the same level of requirements for the environmental envelope as specified for the electrical connector in Appendix A, as applicable.

5.3.6 Store Receptacle Connector Orientation and Mounting. Orientation and mounting applies to both the SSI and Auxiliary Power connectors.

5.3.6.1 Top Mounting. With the connector positioned such that the longitudinal axis (the axis that traverses the connector from front to back of the connector) is in the vertical plane and the connector face is facing upward, the master keyway shall be in the forward position on the store longitudinal axis (See Figure 5).

5.3.6.2 Rear Mounting. With the connector positioned such that the longitudinal axis is in the horizontal plane and the connector face is facing rearward (aft), the master keyway shall be in the up position on the store vertical axis (See Figure 5).

5.3.6.3 Mounting Alignment and Integrity. The store receptacle mounting shall be designed to prevent the master keyway from becoming misaligned with the predetermined position. The retaining force of the mounting shall be compatible with the lanyard release force.

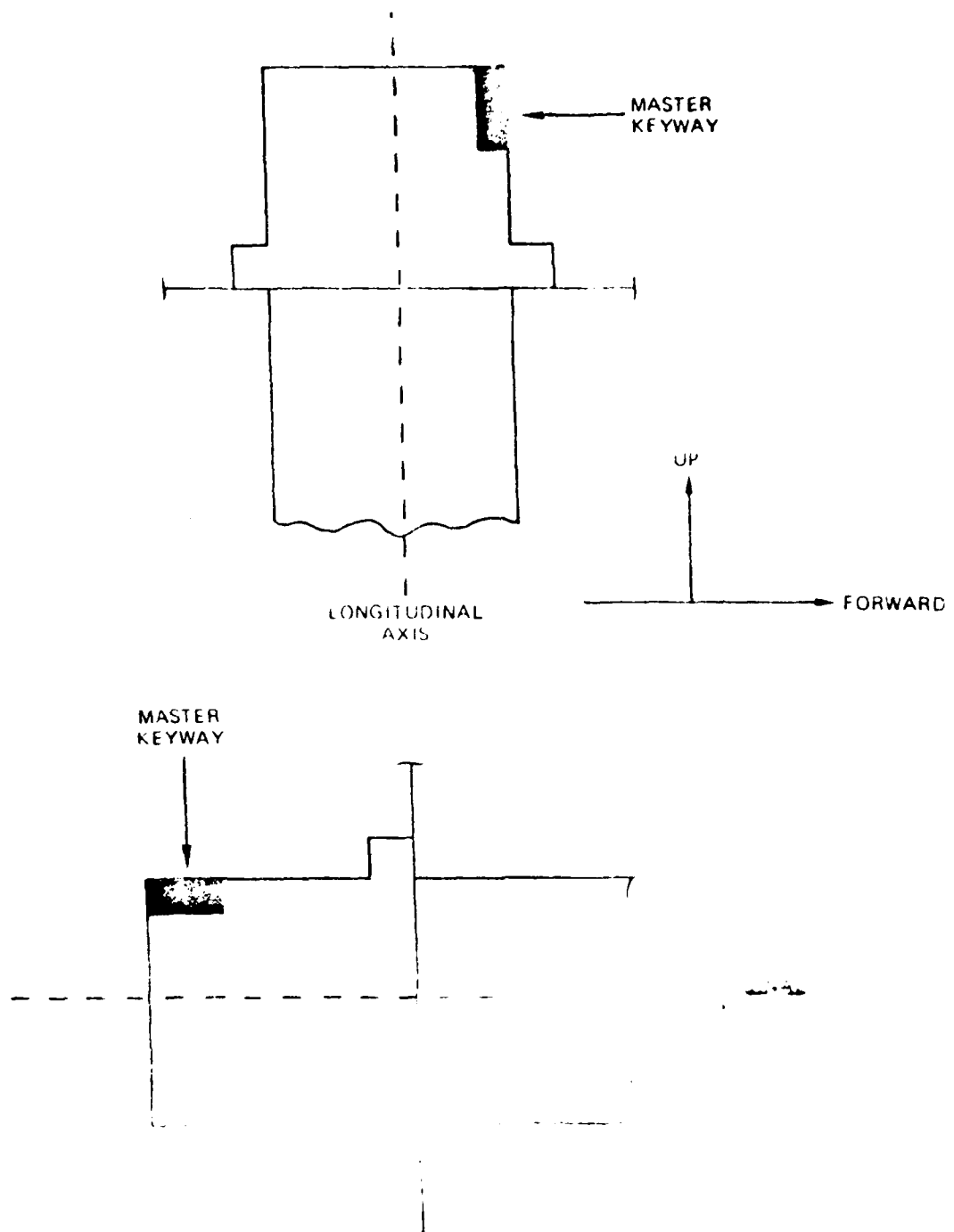


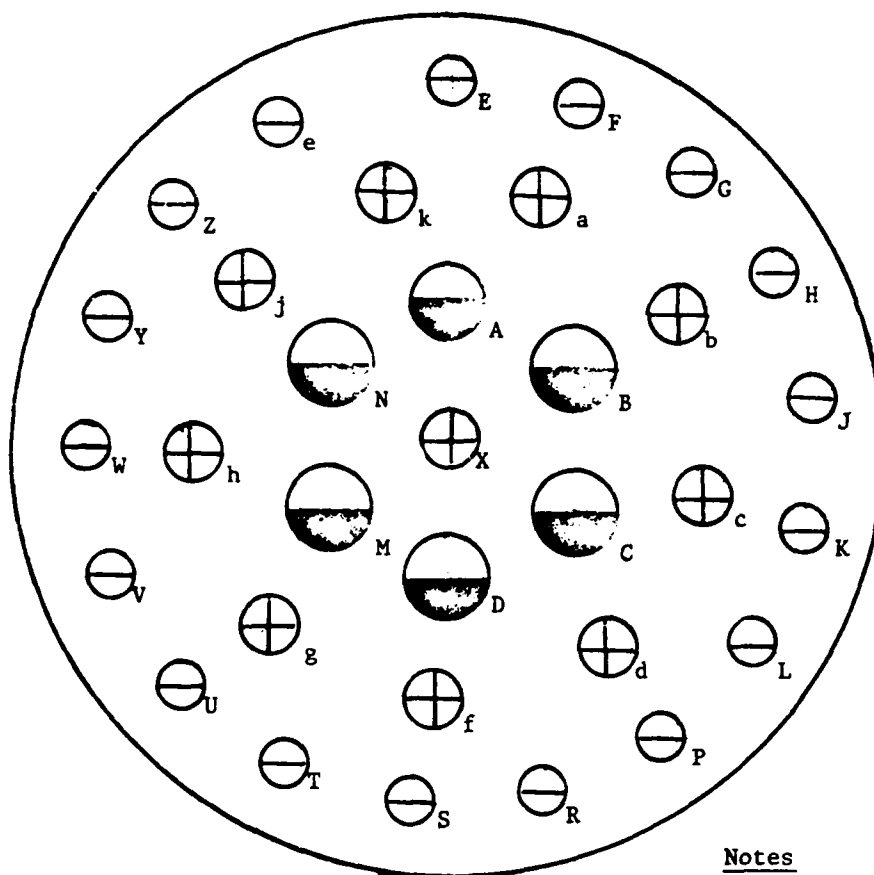
Fig. 5. Master Keyway position

5.3.6.4 Receptacle Position. The location of the receptacle connector shall be in accordance with MIL-A-8591 and NATO Standardization Agreement, STANAG 3558, for lug mounted stores. The location for rail-launched stores is to be determined.

5.3.7 Insert Arrangement. The final insert dimensional geometry is to be determined through a detailed analysis of signal classes (sensitive to pulse, RF) and the relationship of interference, sensitivity, and temperature loading factors between the respective circuits. Figures 5a and 6b depict, in non-dimensional geometry, a nominal arrangement of contacts for the electrical signal sets contained in the insert. The connector shell size is a function of the insert diameter. It is estimated that this shell size will be 11 to 25. The number and type of spare contacts are to be determined.

5.3.8 Exposed Receptacle Pins. The exposed pins in the store receptacle shall be shielded to prevent interference coupling after umbilical connection. The store design shall incorporate features to shield the receptacle(s) from EMI to the same level as the store.

5.3.9 Interface Definition. This section of the standard defines the structured data transfer procedure which is consistent with the requirements set forth in the digital command/response, time division multiplexing standard (MIL-STD-1553), and provides for standardization of information transfer (logical interface) across the aircraft/store interface. Included will be interface message traffic and information coding format. Details of this portion of the standard are TBD.



Notes

1. Refer to Table 1 for pin function.
2. This pin arrangement is for illustrative purposes only.
3. Spare pins are not shown.

Gauge

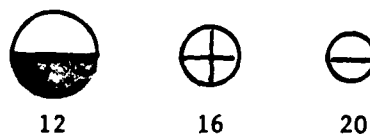
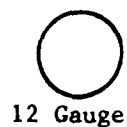
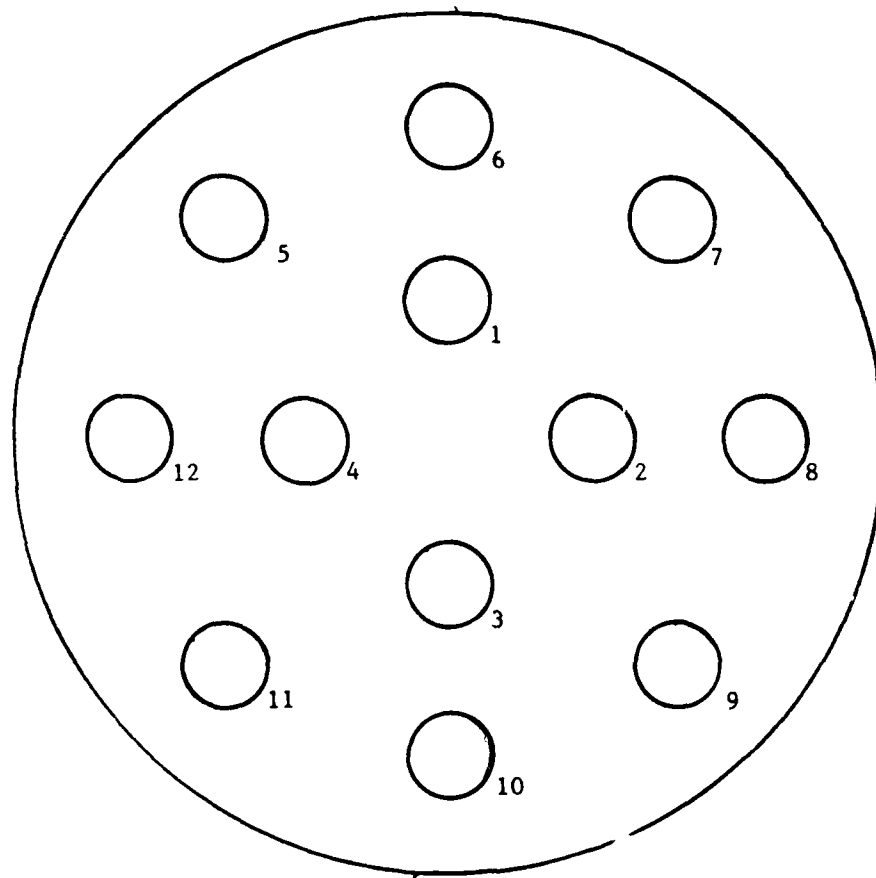


Figure 6a. SSI Connector - insert arrangement



Notes

1. Refer to Table 1 for pin function.
2. This pin arrangement is for illustrative purposes only.
3. Spare pins are not shown.

Figure 6b. Auxiliary Power Connector - insert arrangement

TABLE 1

ELECTRICAL SIGNAL SETS

SSI CONNECTOR

AUXILIARY POWER CONNECTOR

TABLE 1. Electrical Signal Set - SSI Connector.

PIN:	A
TITLE:	High Bandwidth Line 1
FUNCTION:	Provide signal path for video or any high bandwidth
SOURCE:	Aircraft/Store
DESTINATION:	Aircraft/Store
RETURN:	Coaxial shield
CONTACT:	Size 12 coaxial
CHARACTERISTICS:	Type of signal: Video or any high bandwidth
LOAD:	See remarks
REMARKS:	This line can be used to carry video or any high bandwidth signals between aircraft and store. The exact function of this line shall be determined by the type of store being carried and shall be controlled by the aircraft stores management system. When used for video, parameters shall be in accordance with EIA Standard RS-330 with the following exceptions: 1. α (sync pulse amplitude) will be $.8 \pm .08$ volts. 2. The standard load impedance of the video sensor shall be $93 \text{ ohms} \pm 10 \text{ percent}$. 3. Video shall be 525 lines, or 875 lines in accordance with EIA Standard RS-343A

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	B
TITLE:	High Bandwidth Line 2
FUNCTION:	Provide signal path for video or any high bandwidth signal to/from aircraft/store.
SOURCE:	Aircraft/Store
DESTINATION:	Aircraft/Store
RETURN:	Coaxial shield
CONTACT:	Size 12 coaxial
CHARACTERISTICS:	Type of signal: Video or any high bandwidth
LOAD:	See remarks
REMARKS:	This line can be used to carry video or any high bandwidth signals between aircraft and store. The exact function of this line shall be determined by the type of store being carried and shall be controlled by the aircraft stores management system. When used for video, parameters shall be in accordance with EIA Standard RS-330 with the following exceptions: 1. α (sync pulse amplitude) will be $.8 \pm .08$ volts. 2. The standard load impedance of the video sensor shall be 93 ohms ± 10 percent. 3. Video shall be 525 lines, or 875 lines in accordance with EIA Standard RS-343A.

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	C
TITLE:	High Bandwidth Line 3
FUNCTION:	Provide signal path for video or any high bandwidth
SOURCE:	Aircraft/Store
DESTINATION:	Aircraft/Store
RETURN:	Coaxial shield
CONTACT:	Size 12 coaxial
CHARACTERISTICS:	Type of signal: Video or any high bandwidth
LOAD:	See remarks
REMARKS:	This line can be used to carry video or any high bandwidth signals between aircraft and store. The exact function of this line shall be determined by the type of store being carried and shall be controlled by the aircraft stores management system. When used for video, parameters shall be in accordance with EIA Standard RS-330 with the following exceptions; 1. α (sync pulse amplitude) will be $.8 \pm .08$ volts. 2. The standard load impedance of the video sensor shall be 93 ohms ± 10 percent. 3. Video shall be 525 lines, or 875 lines in accordance with EIA Standard RS-343A.

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	D
TITLE:	High Bandwidth Line 4
FUNCTION:	Provide signal path for video or any high bandwidth signal to/from aircraft/store.
SOURCE:	Aircraft/Store
DESTINATION:	Aircraft/Store
RETURN:	Coaxial shield
CONTACT:	Size 12 coaxial
CHARACTERISTICS:	Type of signal: Video or any high bandwidth
LOAD:	See remarks
REMARKS:	This line can be used to carry video or any high bandwidth signals between aircraft and store. The exact function of this line shall be determined by the type of store being carried and shall be controlled by the aircraft stores management system. When used for video, parameters shall be in accordance with EIA Standard RS-330 with the following exceptions: 1. α (sync pulse amplitude) will be $.8 \pm .08$ volts. 2. The standard load impedance of the video sensor shall be 93 ohms ± 10 percent. 3. Video shall be 525 lines, or 875 lines in accordance with EIA Standard RS-343A.

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	E
TITLE:	Store Audio
FUNCTION:	Target presence and lock-on indicator to pilot
SOURCE:	Store
DESTINATION:	Aircraft
RETURN:	Power 1 Return, pin H
CONTACT:	Size 20
CHARACTERISTICS:	400 to 2000 Hertz amplitude modulated carrier. Zero to 30 volts RMS
LOAD:	10K - 20K ohms
REMARKS:	Store audio output to indicate target presence and/or lock-on.

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	F
TITLE:	Dat Bus A(+)
FUNCTION:	Transmits serial digital multiplex command, control, and status data between the aircraft and the store
SOURCE:	Aircraft
DESTINATION:	Bus interface electronics in store
RETURN:	Data Bus A(-), pin G
CONTACT:	Size 20
CHARACTERISTICS:	Bidirectional signal, Manchester biphase IAW MIL-STD-1553
LOAD:	To be determined
REMARKS:	See 5.2.1. Part of a 2-wire twisted shielded cable. Shield carried through connector. Reference pins G and H.

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	G
TITLE:	Data Bus A(-)
FUNCTION:	Transmits serial digital multiplex command, control, and status data between the aircraft and the store
SOURCE:	Aircraft
DESTINATION:	Bus interface electronics in store
RETURN:	Data Bus A(+), pin F
CONTACT:	Size 20
CHARACTERISTICS:	Ridirectional signal, Manchester biphase IAW MIL-STD-1553
LOAD:	To be determined
REMARKS:	See 5.2.1. Part of a 2-wire twisted shielded cable. Shield carried through connector. Reference pins F and H

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	H
TITLE:	Bus A Shield
FUNCTION:	Protects Data Bus A from spurious signals
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	Not applicable
CONTACT:	Size 20
CHARACTERISTICS:	Multiplex data shield 0 VDC
LOAD:	Not applicable
REMARKS:	Part of a 2-wire twisted shielded cable. Reference pins F and G

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	J
TITLE:	Data Bus(+)
FUNCTION:	Transmits serial digital multiplex command, control, and status data between the aircraft and the store
SOURCE:	Aircraft
DESTINATION:	Bus interface electronics in store
RETURN:	Data Bus B(-), pin K
CONTACT:	Size 20
CHARACTERISTICS:	Bidirectional signal, Manchester biphase IAW MIL-STD-1553
LOAD:	To be determined
REMARKS:	See 5.2.1. Part of a 2-wire twisted shielded cable. Shield carried through connector. Reference pins K and L

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	K
TITLE:	Data Bus B(-)
FUNCTION:	Transmits serial digital multiplex command, control, and status data between the aircraft and the store
SOURCE:	Aircraft
DESTINATION:	Bus interface electronics in store
RETURN:	Data Bus B(+), pin J
CONTACT:	Size 20
CHARACTERISTICS:	Bidirectional signal, Manchester biphase IAW MIL-STD-1553
LOAD:	To be determined
REMARKS:	See 5.2.1. Part of a 2-wire twisted shielded cable. Shield carried through connector. Reference pins J and L

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	L
TITLE:	Bus B Shield
FUNCTION:	Protects Data Bus B from spurious signals
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	Not applicable
CONTACT:	Size 20
CHARACTERISTICS:	Multiple data shield 0 VDC
LOAD:	Not applicable
REMARKS:	Part of a 2-wire twisted shielded cable. Reference pins J and K

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	M
TITLE:	Fiber Optic Data Bus A
FUNCTION:	Transmits serial digital multiplex command, control, and status data between the aircraft and the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	Not applicable
CONTACT:	Size 12 contact for fiber optic cable
CHARACTERISTICS:	To be determined
LOAD:	Not applicable
REMARKS:	Growth provision for standard interface This pin will not be used until fiber optic communications techniques are further developed. Fiber optic communication may take place through a separate advanced interface connector.

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	N
TITLE:	Fiber Optic Data Bus B
FUNCTION:	Transmits serial digital multiplex command, control, and status data between the aircraft and the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	Not applicable
CONTACT:	Size 12 contact for fiber optic cable
CHARACTERISTICS:	To be determined
LOAD:	Not applicable
REMARKS:	Growth provision for standard interface This pin will not be used until fiber optic communications techniques are further developed. Fiber optic communication may take place through a separate advanced interface connector.

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	P
TITLE:	Address Line 1 (most significant bit - MSB)
FUNCTION:	Transmits store digital data bus identification address bit from aircraft to bus interface electronic system
SOURCE:	Aircraft stores management system
DESTINATION:	Bus interface electronics in store
RETURN:	Address, Return Pin W
CONTACT:	Size 20
CHARACTERISTICS:	Logic 1 = Open Circuit (≥ 100 K ohms) Logic 0 = Short Circuit connected to Pin W (≤ 10 ohms)
LOAD:	Not applicable
REMARKS	Reference pins R, S, T, U, V, and W

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	R
TITLE:	Address Line 2 (MSB-1)
FUNCTION:	Transmits store digital data bus identification address bit from aircraft to bus interface electronic system
SOURCE:	Aircraft stores management system
DESTINATION:	Bus interface electronics in store
RETURN:	Address, Return Pin W
CONTACT:	Size 20
CHARACTERISTICS:	Logic 1 = Open Circuit (≥ 100 K ohms) Logic 0 = Short-Circuit connected to Pin W (≤ 10 ohms)
LOAD:	Not applicable
REMARKS:	Reference pins P, S, T, U, V, and W

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	S
TITLE:	Address Line 3 (MSB-2)
FUNCTION:	Transmits store digital data bus identification address bit from aircraft to bus interface electronic system
SOURCE:	Aircraft stores management system
DESTINATION:	Bus interface electronics in store
RETURN:	Address, Return Pin W
CONTACT:	Size 20
CHARACTERISTICS:	Logic 1 = Open Circuit (≥ 100 K ohms) Logic 0 = Short Circuit connected to Pin W (≤ 10 ohms)
LOAD:	Not applicable
REMARKS:	Reference pins P, R, T, U, V, and W

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	T
TITLE:	Address Line 4 (MSB-4)
FUNCTION:	Transmits store digital data bus identification address bit from aircraft to bus interface electronic system
SOURCE:	Aircraft stores management system
DESTINATION:	Store/bus interface electronics
RETURN:	Address, Return Pin W
CONTACT:	Size 20
CHARACTERISTICS:	Logic 1 = Open Circuit (≥ 100 K ohms) Logic 0 = Short Circuit connected to Pin W (≤ 10 ohms)
LOAD:	Not applicable
REMARKS:	Reference pins P, R, S, U, V, and W

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	U
TITLE:	Address Line 5 (least significant bit - LSB)
FUNCTION:	Transmits store digital data bus identification address bit from aircraft to bus interface electronic system
SOURCE:	Aircraft stores management system
DESTINATION:	Bus interface electronics in store
RETURN:	Address, Return Pin W
CONTACT:	Size 20
CHARACTERISTICS:	Logic 1 = Open Circuit (≥ 100 K ohms) Logic 0 = Short Circuit connected to Pin W (≤ 10 ohms)
LOAD:	Not applicable
REMARKS:	Reference pins P, R, S, T, V, and W

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	V
TITLE:	Parity Line
FUNCTION:	Provides store digital data bus identification address parity bit from aircraft to bus interface electronics system
SOURCE:	Aircraft stores management system
DESTINATION:	Bus interface electronics in store
RETURN:	Address, Return Pin W
CONTACT:	Size 20
CHARACTERISTICS:	Logic 1 = Open Circuit (≥ 100 K ohms) Logic 0 = Short Circuit (≤ 10 ohms connected to Pin Y)
REMARKS:	Reference pins P, R, S, T, U, and W. This line will be set to a logical 1 when the sum of the bits on the 5 address lines is even. This line will be set to a logical 0 when the sum of the bits on the other 5 address lines is odd (ODD PARITY).

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	W
TITLE:	Address Line Return
FUNCTION:	Provide a return path between aircraft and bus interface electronics for the address lines.
SOURCE:	Store
DESTINATION:	Bus interface electronics in store
RETURN:	Not applicable
CONTACT:	Size 20
CHARACTERISTICS:	
LOAD:	Not applicable
REMARKS:	Reference pins P, R, S, T, U, and V. This line is the address return for the address lines. It shall be isolated from ground and not connected to store structure.

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	X
TITLE:	Emergency Jettison
FUNCTION:	Operates from aircraft emergency jettison bus. Provides mechanically and electrically isolated jettison signal to store interface
SOURCE:	Aircraft emergency jettison controller
DESTINATION:	Store jettison activator system
RETURN:	Power 2 Rtn, pin j
CONTACT:	Size 16
CHARACTERISTICS:	Discrete signal 28 VDC: Jettison store order Open: Jettison not ordered
LOAD:	Maximum current is 10 amperes
REMARKS:	High-priority aircraft command. Requires dedicated hardwire circuit

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PINS:	Y, Z
TITLE:	Store Present Interlock
FUNCTION:	Indicates store-present/store-gone status to aircraft
SOURCE:	Aircraft (store-present indication), pin Y
DESTINATION:	Store
RETURN:	Aircraft, pin Z
CONTACT:	Size 20
CHARACTERISTICS:	Pin Y to pin Z shortened indicates store present (≤ 10 ohms) Pin Y to pin Z open indicates store gone (≥ 100 K ohms)
LOAD:	Not applicable
REMARKS:	Positive store disconnect/release indicator to aircraft. See figure 3a.

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	a
TITLE:	115 VAC 400 HZ PHASE A
FUNCTION:	Provides one phase of the 115 volt, 400 Hertz, AC power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	115 VAC 400 HZ RTN, pin d
CONTACT:	Size 16
CHARACTERISTICS:	Power Signal: 115 VAC, 400 Hertz, Phase A, 4-wire, wye-connected power
LOAD:	Maximum sustained current is 10 amperes
REMARKS:	Aircraft power shall be in accordance with MIL-STD-704. The aircraft shall provide interface fault protection. Reference pins b, c, and d

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	b
TITLE:	115 VAC 400 HZ PHASE B
FUNCTION:	Provides one phase of the 115 volt, 400 hertz ac power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	115 VAC 400 HZ RTN, pin d
CONTACT:	Size 16
CHARACTERISTICS:	Power Signal: 115 VAC, 400 Hertz, Phase B, 4-wire, wye-connected power
LOAD:	Maximum sustained current is 10 amperes
REMARKS:	This power shall be in accordance with MIL-STD-704. The aircraft shall provide interface fault protection. Reference pins a, c, and d

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	c
TITLE:	115 VAC 400 HZ PHASE C
FUNCTION:	Provides one phase of the 115 volt, 400 hertz, power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	115 VAC 400 HZ RTN, pin d
CONTACT:	Size 16
CHARACTERISTICS:	Power Signal: 115 VAC, 400 Hertz, Phase C 4-wire, wye-connected power
LOAD:	Maximum sustained current is 10 amperes
REMARKS:	This power shall be in accordance with MIL-STD-704. The aircraft shall provide interface fault protection. Reference pins a, b, and d

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	d
TITLE:	115 VAC 400 HZ RTN
FUNCTION:	Provides a neutral and a return path for unbalanced loads connected to the 115 VAC, 400 hertz, three-phase four-wire, wye-connected power supplied by the aircraft
SOURCE:	Store
DESTINATION:	Aircraft
RETURN:	This is the ac power return signal
CONTACT:	Size 16
CHARACTERISTICS:	Power Signal: Neutral-0 VAC
LOAD:	Not applicable
REMARKS:	Phases A, B, and C are nominally 120 degrees out of phase with each other; therefore, the current in the neutral wire (return) is not equal to the simple sum of the maximum phase currents. Reference pins a, b, and c

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	e
TITLE:	Initial Store Power
FUNCTION:	Activates bus interface electronic system in store to enable store operation on aircraft digital data buses
SOURCE:	Aircraft 28 VDC power bus
DESTINATION:	Bus interface electronic system in store
RETURN:	POWER 1 RTN, pin h
CONTACT:	Size 20
CHARACTERISTICS:	+28 VDC (nominal) applied when store electrical bus interface activation is required
LOAD:	1.00 ampere maximum
REMARKS:	Power shall be supplied per MIL-STD-704

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	f
NAME:	28 VDC POWER 1
FUNCTION:	Supplies continuous 28 VDC aircraft power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	POWER 1 RTN, pin h
CONTACT:	Size 16
CHARACTERISTICS:	28 VDC power
LOAD:	Maximum sustained current is 10 amperes
REMARKS:	The 28 VDC power shall be in accordance with MIL-STD-704 except that the aircraft shall be capable of supplying 22 VDC minimum to the interface at the above specified current level. The aircraft shall provide interface fault protection

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	8
TITLE:	28 VDC POWER 2
FUNCTION:	Supplies continuous 28 VDC aircraft power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	POWER 2 RTN, pin j
CONTACT:	Size 16
CHARACTERISTICS:	28 VDC power
LOAD:	Maximum sustained current is 10 amperes
REMARKS:	The 28 VDC power shall be in accordance with MIL-STD-704 except that the aircraft shall be capable of supplying 22 VDC minimum to the interface at the above specified current level. The aircraft shall provide interface fault protection

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	h
TITLE:	POWER 1 RTN
FUNCTION:	Provides a return path between the aircraft and store for 28 VDC Power 1
SOURCE:	Aircraft and store
DESTINATION:	Store and aircraft
RETURN:	No applicable
CONTACT:	Size 16
CHARACTERISTICS:	0 VDC power
LOAD:	Maximum sustained current in the ground return line is 11 amperes
REMARKS:	Shall be isolated from ground and not connected to weapon structure. Reference pins E and e

TABLE 1. Electrical Signal Set - SSI Connector. - Continued

PIN:	j
TITLE:	POWER 2 RTN
FUNCTION:	Provides a return path between the aircraft and store for 28 VDC Power 2
SOURCE:	Aircraft and store
DESTINATION:	Store and aircraft
RETURN:	Not applicable
CONTACT:	Size 16
CHARACTERISTICS:	0 VDC power
LOAD:	Maximum sustained current in the ground return line is 10 amperes
REMARKS:	Shall be isolated from ground and not connected to structure in weapon. Reference pins x and g

TABLE 1. Electrical Signal Set - SSI Connector. - Concluded

PIN:	k
TITLE:	Structure Ground
FUNCTION:	Provides ground safety interconnect between aircraft and store structure ground
SOURCE:	Aircraft frame ground
DESTINATION:	Store frame ground
RETURN:	Not applicable
CONTACT:	Size 16
CHARACTERISTICS:	0 VDC
LOAD:	Not applicable
REMARKS:	Shall not be used for signal or power return path

TABLE 1. Electrical Signal Set - Auxiliary Power Connector.

PIN:	1
TITLE:	115 VAC 400 Hz PHASE A
FUNCTION:	Provides one phase of the 115 volt, 400 Hertz, AC power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	115 volt 400 Hz RTN, pin 7 and 8
CONTACT:	Size 12
CHARACTERISTICS:	Power Signal: 115 VAC, 400 Hz, Phase A 4 - wire, wye-connected power
LOAD:	Maximum sustained current is 23 amperes.
REMARKS:	Aircraft power shall be in accordance with MIL-STD-704. The aircraft shall provide interface fault protection. This pin supplies half of the available power of phase A. Reference pins 1 through 8.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	2
TITLE:	115 VAC 400 Hz PHASE A
FUNCTION:	Provides one phase of the 115 volt, 400 Hertz, AC power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	115 volt 400 Hz RTN, pin 7 and 8
CONTACT:	Size 12
CHARACTERISTICS:	Power Signal: 115 VAC, 400 Hz, Phase A, 4 - wire, wye-connected power
LOAD:	Maximum sustained current is 23 amperes.
REMARKS:	Aircraft power shall be in accordance with MIL-STD-704. The aircraft shall provide interface fault protection. This pin supplies half of the available power of phase A. Reference pins 1 through 8.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	3
TITLE:	115 VAC 400 Hz PHASE B
FUNCTION:	Provides one phase of the 115 volt, 400 Hertz, AC power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	115 volt 400 Hz RTN, pin 7 and 8
CONTACT:	Size 12
CHARACTERISTICS:	Power Signal: 115 VAC, 400 Hz, Phase A, 4 - wire, wye-connected power
LOAD:	Maximum sustained current is 23 amperes.
REMARKS:	Aircraft power shall be in accordance with MIL-STD-704. The aircraft shall provide interface fault protection. This pin supplies half of the available power of phase B. Reference pins 1 through 8.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	4
TITLE:	115 VAC 400 Hz PHASE B
FUNCTION:	Provides one phase of the 115 volt, 400 Hertz, AC power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	115 volt 400 Hz RTN, pin 7 and 8
CONTACT:	Size 12
CHARACTERISTICS:	Power Signal: 115 VAC, 400 Hz, Phase A, 4 - wire, wye-connected power
LOAD:	Maximum sustained current is 23 amperes.
REMARKS:	Aircraft power shall be in accordance with MIL-STD-704. The aircraft shall provide interface fault protection. This pin supplies half of the available power of phase B. Reference pins 1 through 8.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	5
TITLE:	115 VAC 400 Hz PHASE C
FUNCTION:	Provides one phase of the 115 volt, 400 Hertz, AC power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	115 volt 400 Hz RTN, pin 7 and 8
CONTACT:	Size 12
CHARACTERISTICS:	Power Signal: 115 VAC, 400 Hz, Phase A, 4 - wire, wye-connected power
LOAD:	Maximum sustained current is 23 amperes.
REMARKS:	Aircraft power shall be in accordance with MIL-STD-704. The aircraft shall provide interface fault protection. This pin supplies half of the available power of phase C. Reference pins 1 through 8.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	6
TITLE:	115 VAC 400 Hz PHASE C
FUNCTION:	Provides one phase of the 115 volt, 400 Hertz, AC power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	115 volt 400 Hz RTN, pin 7 and 8
CONTACT:	Size 12
CHARACTERISTICS:	Power Signal: 115 VAC, 400 Hz, Phase A 4 - wire, wye-connected power
LOAD:	Maximum sustained current is 23 amperes.
REMARKS:	Aircraft power shall be in accordance with MIL-STD-704. The aircraft shall provide interface fault protection. This pin supplies half of the available power of Phase C. Reference pins 1 through 8.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	7
TITLE:	115 VAC 400 Hz RTN
FUNCTION:	Provides a neutral and a return path for unbalanced loads connected to the 115 VAC, 400 Hertz, three-phase four wire, wye-connected power supplied by the aircraft
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	This is the AC power return signal
CONTACT:	Size 12
CHARACTERISTICS:	Power Signal: Neutral - 0 VAC
LOAD:	Not applicable
REMARKS:	Phases A, B and C are nominally 120 degrees out of phase with each other; therefore, the current in the neutral wire (return) is not equal to the simple sum of the maximum phase currents. This pin is half of the return line. Reference pins 1 through 8.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector - Continued

PIN:	8
TITLE:	115 VAC 400 Hz RTN
FUNCTION:	Provides a neutral and a return path for unbalanced loads connected to the 115 VAC, 400 Hertz, three-phase four wire, wye-connected power supplied by the aircraft
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	This is the AC power return signal
CONTACT:	Size 12
CHARACTERISTICS:	Power Signal: Neutral - 0 VAC
LOAD:	Not applicable
REMARKS:	Phases A, B and C are nominally 120 degrees out of phase with each other; therefore, the current in the neutral wire (return) is not equal to the simple sum of the maximum phase currents. This pin is half of the return line. Reference pins 1 through 8.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	9
TITLE:	28 VDC POWER BUS C
FUNCTION:	Supplies continuous 28 VDC aircraft power to the store.
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	Power return, pins 11 and 12
CONTACT:	Size 12
CHARACTERISTICS:	28 VDC power
LOAD:	Maximum sustained current is 23 amperes
REMARKS:	The 28 VDC power shall be in accordance with MIL-STD-704 except that the aircraft shall be capable of supplying 22 VDC minimum to the interface at the above specified level. The aircraft shall provide interface fault protection. This pin is half of the DC lines. Reference pins 9 through 12.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	10
TITLE:	28 VDC POWER BUS C
FUNCTION:	Supplies continuous 28 VDC aircraft power to the store
SOURCE:	Aircraft
DESTINATION:	Store
RETURN:	Power return, pins 11 and 12
CONTACT:	Size 12
CHARACTERISTICS:	28 VDC power
LOAD:	Maximum sustained current is 23 amperes
REMARKS:	The 28 VDC power shall be in accordance with MIL-STD-704 except that the aircraft shall be capable of supplying 22 VDC minimum to the interface at the above specified level. The aircraft shall provide interface fault protection. This pin is half of the DC lines. Reference pins 9 through 12.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	11
TITLE:	POWER RETURN C
FUNCTION:	Provides a return path between the aircraft and store for 28 VDC Power Bus C
SOURCE:	Store
DESTINATION:	Aircraft
RETURN:	This is a DC power return line
CONTACT:	Size 12
CHARACTERISTICS	0 VDC power
LOAD:	Maximum sustained current in the ground return line is 23 amperes.
REMARKS:	Shall be isolated from ground and not connected to structure in weapon. This pin is one half of the DC return lines. Reference pins 9 through 12.

TABLE 1. Electrical Signal Set - Auxiliary Power Connector. - Continued

PIN:	12
TITLE:	POWER RETURN C
FUNCTION:	Provides a return path between the aircraft and store for 28 VDC Power Bus C
SOURCE:	Store
DESTINATION:	Aircraft
RETURN:	This is a DC power return line
CONTACT:	Size 12
CHARACTERISTICS:	0 VDC power
LOAD:	Maximum sustained current in the ground return line is 23 amperes.
REMARKS:	Shall be isolated from ground and not connected to structure in weapon. This pin is one half of the DC return lines. Reference pins 9 through 12.

APPENDIX A

6. ELECTRICAL CONNECTOR REQUIREMENTS

6.1 Introduction. This section prescribes detail electrical connector physical requirements for the standard aircraft/store interface. Specific physical and operating requirements for the standard interface connector are given. The following requirements shall be in accordance with MIL-C-38999, or subsequent revisions. The following material has been reproduced from MIL-C-38999 to minimize the need for additional documents to obtain the requirements. This material applies to Series IV connectors only. Requirements not specific to MIL-C-38999 are also presented (where possible) which indicate a reasonable alternative to a design in preference to nothing at all or a "to be determined". The underlined paragraph number, e.g. 4.8.2, is the appropriate test paragraph of MIL-C-38999 and the parenthetical number (3.5.6) is the requirement paragraph of MIL-C-38999.

6.2 Definitions.

6.2.1 Electrical Connector Terminology. See Section 3 of MIL-STD-1353 for standard electrical connector terminology definitions.

6.2.2 Requirements. Requirements described in the following sections are of two types. The first, Section 6.3 (General) are written in general terms to address major connector characteristics definitely required for the standard interface applications. Characteristic descriptions have been derived from known technology capabilities and/or an analysis of necessary connector features. The second type, contained in 6.4 (Detailed), and referenced to MIL-C-38999, are considered to be a boiler plate requirements applicable to and contained in all airborne connector specifications. The standard is not intended to form a basis for the formulation of revisions to existing specifications.

6.3 General Requirements.

6.3.1 Design and Construction. (3.4) Connectors and accessories shall be designed and constructed to withstand normal handling incident to installation and maintenance in service. Connector receptacle interchangeability control dimensions and accessory interface dimensions shall be as specified on Figure 1. All accessories (plugs) designed to be used with MIL-C-38999 connectors must conform to Figure 2.

6.3.2 Types.

- a. Receptacle, aircraft shall contain socket contacts.
- b. Plug, aircraft umbilical to store shall contain socket contacts.

6.3.3 Finish. The finish shall be electroless nickle coating (conductive) -65°C to $+200^{\circ}\text{C}$, followed by cadmium plate 0.0001 inch (0.003mm) minimum in accordance with QQ-P-416, Type II.

6.3.4 Interchangeability and Intermateability. The connectors (Figures 1, 2 and 3) shall be completely interchangeable and intermateable from all vendor sources and production runs. (3.5) All connectors having the same part number shall be completely interchangeable with each other with respect to installation and performance.

6.3.5 Pin Protection. The receptacle connector shall be designed to prevent the plug shell from contacting the receptacle pins under any conditions of mating. This design is commonly referred to as "scoop proof".

6.3.6 Coupling. Connectors shall be coupled to counterpart connectors by means of a breech mechanism (Series IV). The mechanism shall include a means of maintaining the mated connector in full engagement. The coupling ring shall be knurled or fluted to facilitate coupling and shall be captivated. The coupling nuts of all connectors shall have a blue color band in accordance with EIA RS-359, indicating a rear release contact retention system.

6.3.6.1 Ease of Coupling. (3.4.6.1) Counterpart connectors of any arrangement shall be capable of being fully coupled and uncoupled in a normal and accessible location without the use of tools.

6.3.7 Engagement and Locking. Counterpart connectors shall be capable of full engagement and disengagement without the use of tools. Engagement of connectors shall be defined as mated insert faces. For Series IV, complete coupling shall be accomplished by approximately 90° clockwise rotation of the coupling nut, and shall incorporate a positive detent action at both the mated and unmated position, providing an audible and tactile indication of complete coupling, as well as an anti-decoupling force. A red band shall be located on the plug so as to be visible when unmated and fully covered when completely mated.

6.3.8 Polarization of Connector Shells. (3.4.6.3) Polarization of connector shells shall be accomplished by means of five integral keys and suitable matching keyways on the counterpart (See Figures 1 and 2). Polarization shall be accomplished before initial engagement of the coupling ring. During axial engagement, pins shall not touch sockets or the insert face until polarization has been achieved.

6.3.8.1 Pin to Pin Mating Prevention (Series IV only). Series connectors shall be provided with key keyway widths arranged so as to prevent a plug with pin contacts from being mated with a receptacle with pin contacts.

6.3.9 Electrical Continuity. The connectors shall be designed to provide positive electrical continuity between mated shells prior to contact engagement.

6.3.10 Electromagnetic Interface (EMI) Grounding Spring Fingers. EMI grounding spring fingers shall be provided and shall be designed in a manner which will ensure proper engagement of the mating shells and provide electrical contact.

6.3.11 Lanyard. The lanyard shall be mounted to the plug connector (aircraft-to-store 6.3.2.6) such that rotation of the connector for engagement will not shorten the effective length of the lanyard. The design of the lanyard to connector attaching feature shall provide for attaching and removing lanyard cables using the normal compliment of standard shop tools. Design of the lanyard to connector attaching feature shall ensure total integrity of the connector to meet the required lanyard retention force.

6.3.11.1 Lanyard Retention. The lanyard, including all hardware and joints, shall withstand an axial tensile force of 150 pounds minimum.

6.3.11.2 Lanyard Release Force. The lanyard release force shall be no less than 20 pounds and no more than 40 pounds. Direction of lanyard pull shall be at any angle within 30 degrees of the connector longitudinal axis.

6.3.12 Shielding Braid Termination. The plug connector backshell shall provide for shielding braid termination to allow integral bounding of cable envelop shielding to the connector.

6.3.13 Water Sealing. The plug connector backshell shall be designed to ensure complete sealing from entrance of water. A suitable quality assurance test shall be specified to encompass the aircraft/store operational environment.

6.3.14 Electrical Contacts. The following contact types shall be provided as specified, in MIL-C-39029A, or from vendor sources as applicable. The range of size within the types specified shall be compatible with the cavity size in the insert specified for this standard. Contact types of one size shall be completely interchangeable and intermateable within the cavity.

- a. Power Contacts - shielded and unshielded, crimp removable.
- b. Coaxial Contacts
- c. Fiber Optics

6.3.14.1 Contact Sizes. See Table 1, MIL-STD-1760.

6.3.14.2 Insert Arrangement. TBD

6.3.14.3 Connector Shell Size. A size 25 connector shell shall be used for the insert of 6.3.14.2.

6.3.15 Electromagnetic Interference Effectiveness. Unless otherwise stated herein, MIL-STD-461 of current issue, will apply. When tested as specified in paragraph 4.7.27 of MIL-C-38999 of current issue, the EMI shielding effectiveness of mated shells shall not be less than that specified in Table I.

TABLE II. EMI Shielding Effectiveness (Class W)

Frequency MHz	Leakage Attenuation db minimum
.014-100	90
100	90
200	85
300	83
400	81
800	76
1,000	75
1,500	69
2,000	65
3,000	61
4,000	58
6,000	55
10,000	50

6.3.15.1 Electromagnetic Pulse (EMP) Susceptibility. To be determined.

6.3.16 Receptacle Mounting. The most common mounting designs are flange and jam nut.

6.3.16.1 Store Mounting. The type of receptacle mounting design is optional. The type of mounting used shall have a structural strength compatible with the lanyard release force. The type mounting shall be sufficiently rigid to prevent the receptacle turning from a predetermined fixed master polarization key position.

6.3.16.2 Aircraft Mounting. The type of receptacle mounting design shall be a flange. The type mounting shall be sufficiently rigid to prevent the receptacle turning from a predetermined fixed master polarization key position.

6.4 Detailed Requirements.

6.4.1 Materials. (3.3)

6.4.1.1 Metals. Metals shall be of a corrosion-resistant type or shall be plated or treated to resist corrosion.

6.4.1.2 Dissimilar Metals and Compatible Couples. When dissimilar metals are used in intimate contact with each other, protection against electrolysis and corrosion shall be provided. The use of dissimilar metals in contact, which tend toward active electrolytic corrosion, (particularly brass, copper, or steel used in contact with aluminum or aluminum alloy) is not acceptable. However metal plating or metal spraying of dissimilar-base metals to provide similar or suitable abutting surface is permitted. The use of dissimilar metals separated by a suitable insulating material is also permitted. Dissimilar metals and compatible couples are defined in requirement 16 of MIL-STD-454.

6.4.1.3 Hydrolytic Stability. (3.3.1.2) All nonmetallic material shall be selected to meet the hydrolytic resistance requirements specified in requirement 47 of MIL-STD-454.

6.4.2 Components. (3.3.2) Material for specific components of the connector shall be as follows:

6.4.2.1 Class F. (3.3.2.1)

- a. Shell - impact extruded or machined aluminum alloy.
- b. Coupling ring, jam nut, and potting ring - machined aluminum alloy.
- c. Insert (molded) - reinforced epoxy resin or other suitable rigid dielectric material.
- d. Spring fingers - heat treated beryllium copper or corrosion-resistant steel.
- e. Filler compound - RTV silicone conforming to MIL-A-46146.
- f. Gaskets, grommet, and interface seals - silicone or fluorocarbon elastomer.

6.4.2.2 Fungus Resistant. (3.3.3) Material used in the construction of these connectors shall be fungus inert in accordance with requirement 4 of MIL-STD-454.

6.4.2.3 Nonmagnetic Materials. (3.3.4) The relative permeability of the connector assembly shall be less than 2.0 when measured with an indicator conforming to MIL-I-17214.

6.4.3 Insert Design. (3.4.2)

6.4.3.1 Environment Resisting Classes. (3.4.2.1) The entire insert and wire sealing or wire supporting member of the environment resisting assemblies shall be essentially one integral part, designed to provide suitable sealing and support around the wires and be nonremoval. The rigid dielectric shall be one integral piece. The design shall be such as to permit the removal and replacement of individual contacts into their connector inserts with an MS27534 or MS27495 installing/removal tool. The contact locking device shall be contained in the rigid dielectric insert and shall so retain the contacts as to meet the contact retention requirements of this specification, see Figure 4. Inserts shall be secured to prevent rotation. All pin contact inserts shall have a resilient interface seal bonded to the front face in accordance with the applicable standards. Socket insert entry holes and pin "donut" rings shall conform to MIL-C-38999 Figure 5. Sealing to coaxial cable terminated to shielded contacts may be accomplished by means of separate resilient bushings. If separate resilient bushings are required, they shall be furnished with the connector.

6.4.4 Mating Seal. (3.4.3.3) Plugs and receptacles with pin inserts shall have a resilient face with individual pin barriers. The pin barrier projections shall seal in their respective lead-in chambers of the hard face socket insert. The resilient interfacial seal shall provide individual contact seals in the mated condition to ensure circuit isolations between

each contact and contact to shell. The plugs (Series IV) shall incorporate an O-ring peripheral seal.

6.4.5 Shell (3.4.4) Shells, including mounting flanges, shall be one-piece construction and shall be designed to retain their inserts in one position, both axially and with respect to rotation, by mechanical means. Molding shall be used as the retention means for environment resisting connectors. The receptacle shells of crimp contact connectors shall have a blue color band in accordance with EIA RS-359, indicating a rear release contact retention system. The color band shall be located so that it is readily visible to any person servicing a mounted receptacle connector.

6.4.5.1 Spring Fingers. (3.4.4.1) Spring fingers shall be designed to make electrical contact with the mating shell without interfering with proper engagement. The spring shall be retained about the shell periphery. Minimum engagement of spring fingers shall be 0.040 (1.02mm) prior to contact engagement (Series IV).

6.4.5.2 Jam-Nut Mounting Receptacles. (3.4.4.2) Jam-nut mounting receptacles shall be provided with a mounting nut DoD-C-38999/28 Series IV, all with provisions for locking, and an "O" ring MS9021.

6.4.6 Lubrication. (3.4.6.4) The breech mechanism on Series IV connectors may be coated with a suitable dry film lubricant to MIL-L-46010 (non-graphite).

6.4.7 Plating. (3.4.8)

6.4.7.1 Contacts (Crimp). (3.4.8.1) The plating and the plating thickness on crimp contacts shall be as specified in MIL-C-39029.

6.4.7.2 Shell and Accessory Hardware. (3.4.8.2) Unless otherwise specified, the finish on the shells and accessory hardware shall be in accordance with the following designation:

Series IV, Class:

F - Electrically conductive electroless nickel conforming to MIL-C-26074, Class 3 or 4, followed by cadmium plate 0.0001 inch (0.003mm) minimum in accordance with QQ-P-416, Type II. Use of suitable underplate is permissible.

TABLE III. Contact engagement and separation forces.

Mating end size	Initial			After conditioning		
	Minimum separation force (ounces)	Maximum average engagement force (ounces)	Maximum engagement force (ounces)	Minimum separation force (ounces)	Maximum average engagement force (ounces)	Maximum engagement force (ounces)
	Minimum diameter MS3197 pin	Maximum diameter MS3197 pin	Maximum diameter MS3197 pin	Minimum diameter MS3197 pin	Maximum diameter MS3197 pin	Maximum diameter MS3197 pin
12	3	24	30	2.5	29	36
16	2	24	30	1.5	29	36
20	0.7	12	18	0.6	14	22

6.4.8 Contact engagement and separation force. The contact engagement and separation force shall be within the applicable limits specified in Table III.

6.4.9 Thermal shock (all classes except hermetics). (3.8) When tested as specified in 4.7.4, there shall be no damage detrimental to the operation of the connector.

6.4.10 Coupling torque. (3.10) When tested as specified in 4.7.6, the coupling torque for mating and unmating of counterpart connectors shall meet the requirements of Table IV.

TABLE IV. Coupling torque. 1/

Shell size	Maximum engagement and disengagement		Minimum disengagement	
	<u>Pound inch</u>	<u>Newton meters</u>	<u>Pound inch</u>	<u>Newton meters</u>
25	40	4.6	5	0.6

1/ For Series IV connectors with spring fingers, an axial force as specified in Table III must be encountered prior to coupling torque during engagement, and following coupling torque upon disengagement.

6.4.11 Durability. (3.11) When tested as specified in 4.7.7, the connectors shall show no defects detrimental to the operation of the connectors and shall

meet the subsequent test requirements (see 4.4.3).

6.4.12 Altitude immersion (qualification only) (except hermetics). (3.2) When tested as specified in 4.7.8, the mated connector shall meet a minimum insulation resistance of 1,000 and the requirements of dielectric withstanding voltage as specified in (3.14) 50.17.13.

6.4.13 Insulation resistance. (3.13)

6.4.13.1 Insulation resistance at ambient temperature. (3.13.1) When tested as specified in 4.7.9.1, the insulation resistance between any pair of contacts and between any contact and the shell shall be greater than 5,000 megohms. Insulation resistance after altitude immersion shall be 1,000 megohms minimum. Insulation resistance after humidity shall be 100 megohms minimum.

6.4.13.2 Insulation resistance at elevated temperature. (3.13.2) When tested as specified in 4.7.9.2, the insulation resistance between any pair of contacts and between any contact and the shell shall be greater than 1,000 megohms for environment resisting class connectors.

6.4.14 Dielectric withstanding voltage. (3.14) When tested as specified in 4.7.10.1, or 4.7.10.2, connectors shall show no evidence of flashover or breakdown.

6.4.15 Insert retention. (3.15) When tested as specified in 4.7.11, unmated connectors (molded) shall retain their inserts in their proper location in the shell and there shall be no evidence of cracking, breaking separation from the shell, or loosening of parts.

6.4.16 Salt spray (corrosion). (3.16) When tested as specified in 4.7.12, unmated connectors shall show no exposure of base metal due to corrosion which will adversely affect performance.

6.4.17 Electrical engagement. (3.18) When tested as specified in 4.7.14, wired, mated connectors shall provide a minimum of electrical engagement for and (0.050 inch (1.27mm) Series IV.

6.4.18 External bending moment. (3.19) When tested as specified in 4.7.15, connectors shall show no evidence of damage detrimental to their normal operations nor shall there be any interruption of electrical continuity.

6.4.19 Contact retention. (3.23) When tested as specified in 4.7.19, the axial displacement of the contact shall not exceed 0.013 inch (0.30mm). No damage to contacts or inserts shall result.

6.4.20 Altitude-low temperature. When tested as specified in 4.7.20, the connectors shall meet the requirements of the dielectric withstanding voltage at sea level specified in (3.14) and insulation resistance at ambient temperature specified in (3.13.1).

6.4.21 Vibration (qualification only). (3.26) When tested as specified in 4.7.22, there shall be no electrical discontinuity and there shall be no disengagement of the mated connectors, backing off of the coupling mechanism, evidence of cracking, breaking, or loosening of parts.

6.4.22 Shock. (3.27) When tested as specified in 4.7.23, there shall be no electrical discontinuity and there shall be no disengagement of mated connectors, evidence of cracking, breaking, or loosening of parts.

6.4.23 Shell-to-shell conductivity. (3.28) When tested as specified in 4.7.24, the probes shall not puncture or otherwise damage the connector finish and the maximum measured potential drop across assemblies shall be as follows:

- a. Series IV with spring fingers:
 1. Class W - 2.5 millivolt

6.4.24 Humidity. (3.29) When tested as specified in 4.7.25, wired mated connectors shall show no deterioration which will adversely effect performance of connector. Following the test, insulation resistance shall be 100 megohms or greater.

6.4.25 Shell spring finger forces. (3.30) When tested as specified in 4.7.26, the forces necessary to engage and separate EMI plugs with receptacle shells shall be within the values specified in Table V.

TABLE V Shell spring finger forces.

Shell Size	Axial force			
	Series IV			
	Maximum		Minimum	
	Pounds	Newton	Pounds	Newton
24/25	10	44.5	0.5	2.2

6.4.26 Ozone exposure. (3.32) When tested as specified in 4.7.28, the connector shall show no evidence of a cracking of dielectric material or other damage due to ozone exposure that will adversely effect performance.

6.4.27 Fluid immersion. (3.33) When tested as specified in 4.7.29, connectors shall meet the requirements for coupling torque (3.10) and dielectric withstanding voltage (3.14).

6.4.28 Retention system fluid immersion. (3.33.1) When tested as specified

in 4.7.29.1, the insert assemblies shall meet requirements of contact retention (3.23). Effects of the fluids on resilient sealing shall not be a consideration of this test.

6.4.29 Pin contact stability. (3.34) When tested as specified in 4.7.30, the total displacement of a reference point on the contact tip end shall not exceed the amount shown in Table VI.

TABLE VI Pin contact stability.

Contact size	Total displacement		Force	
	Inch	MM	Pounds	Newton
20	0.054	1.37	0.55	2.4
16	0.075	1.91	1.10	4.9
12	0.075	1.91	1.10	4.9

6.4.30 Contact walkout. (3.35) When tested as specified in 4.7.31, contacts shall not become dislodged from their normal position.

6.4.31 Power contacts.

6.4.31.1 Temperature life. (3.37)

6.4.31.1.1 Temperature life with contact loading. (3.37.1) When tested as specified in 4.7.35.1, the contacts shall maintain their specified locations as shown on figure 1 and there shall be no electrical discontinuity.

6.4.31.1.2 Temperature life. (3.37.2) When tested as specified in 4.7.33.2, for 1,000 hours, connectors shall perform satisfactory and pass succeeding tests in the qualification test sequence.

6.4.32 Electrolytic erosion (Series IV). (3.38) When tested as specified in 4.7.34, pin contact shall show no exposure of base metal due to electrolytic erosion. Corrosion deposits shall not be considered as defects.

6.4.33 Firewall. (3.39) Mated connectors shall prevent passing of a flame through the connector for at least 20 minutes when tested in accordance with 4.7.35. During this period there shall be no flame from outgassing or other causes on the end of the connector protected by the firewall. The current specified in 4.7.35, shall be applied for the first five minutes without break in electrical continuity. During the next minute the connector shall draw no more than 2 amperes when a test potential of 100 to 125 Vac at 60 Hz is applied between adjacent contacts and between contacts and the shell. (Class K)

6.4.34 Coaxial and Fiber Optics Contacts. TBD

6.4.35 Marking. (3.43) Connectors and accessories shall be metal or ink stamped with the manufacture's name or trademark, date, code, and the following information, as applicable. Stamping shall be in accordance with MIL-STD-1285 where space permits. Metal stamping shall be accomplished before plating. The following examples are illustrative:

a. Identification.

D38999/40	W	25	31	P	N
Specification	Class	Shell	Insert	Contact	Polarization
sheet No.		size	arrangement	style	(Figures 1, 2, & 3)
					(A letter is required for all positions)

b. Lot number - 000010.

6.4.35.1 Contact location identification. (3.43.1) Contact locations shall be identified as indicated on Figure TBD and the applicable military standard. All positions shall be identified on the front and rear faces of the insert, except where space limitations make this impracticable. Location of contact identifying characters shall be in close proximity to the holes but need not be placed exactly where indicated on the standard.

6.4.36 Workmanship. (3.44) The connector shall be fabricated in a manner such that the criteria for appearance, fit and adherence to specified tolerances are observed. Particular attention shall be given to neatness and thoroughness of marking parts, plating, welding, soldering, riveting, stacking, and bonding. The connectors shall be free from crazing, cracks, voids, pimples, chips, blisters, pinholes, sharp cutting edges, burrs, and other defects that will effect life, serviceability or appearance.

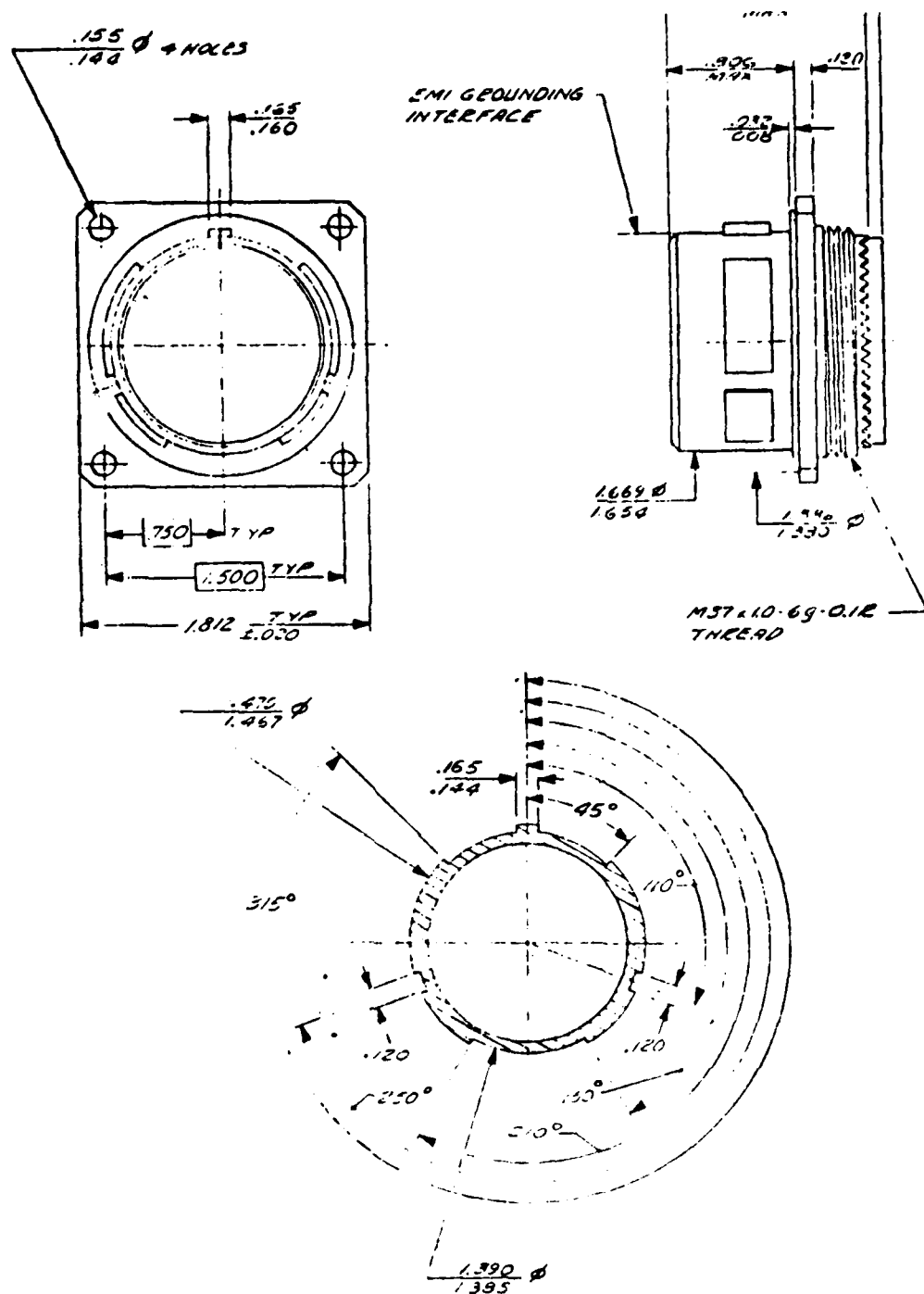


Figure 1. Receptacle, Wall Mount (Cr'mp) (Pin Insert)

A-12

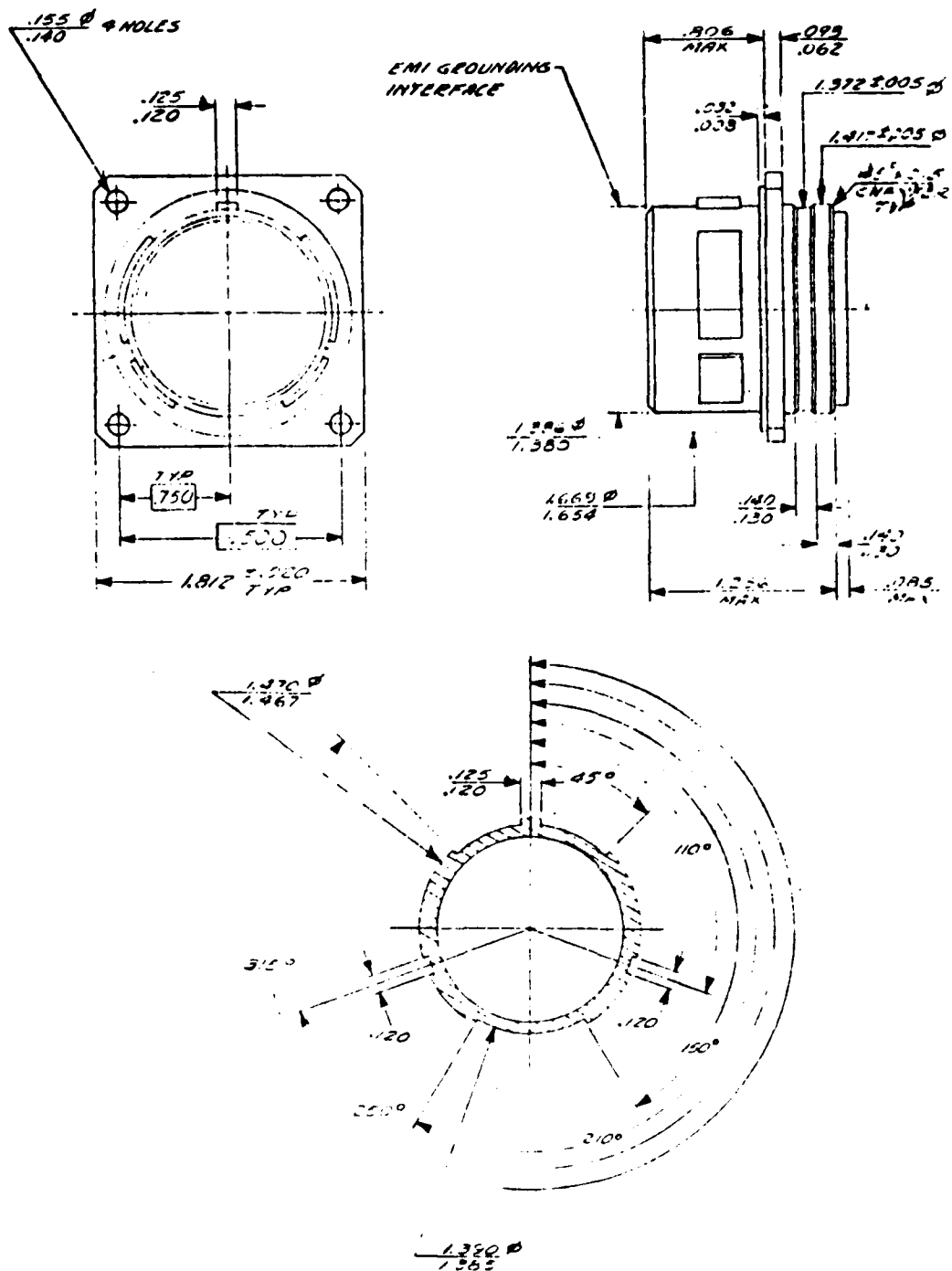


Figure 2. Receptacle, Box (Crimp) (Socket Insert)

A-13

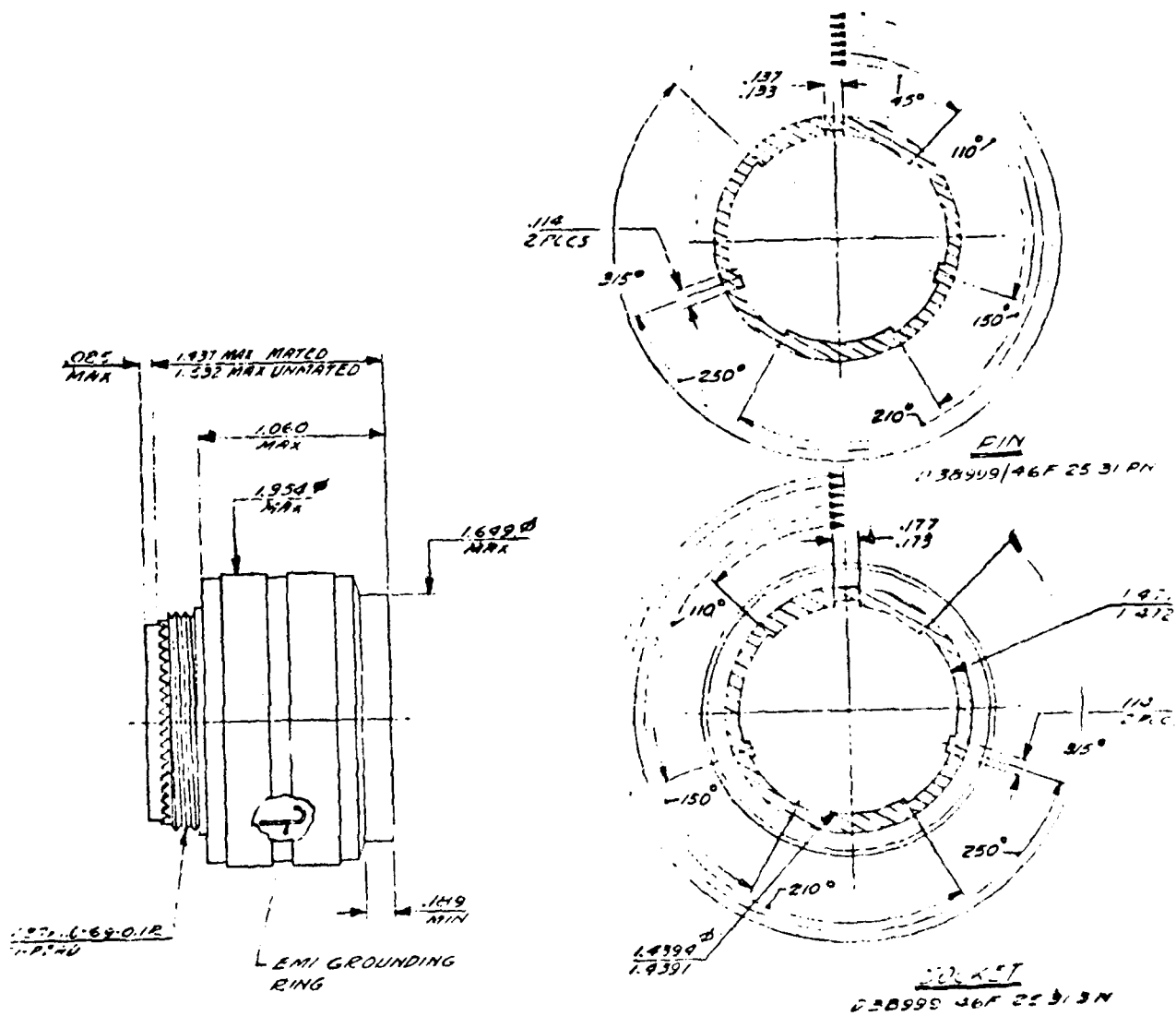


Figure 3. Plug-Straight-Crimp, Pin & Socket

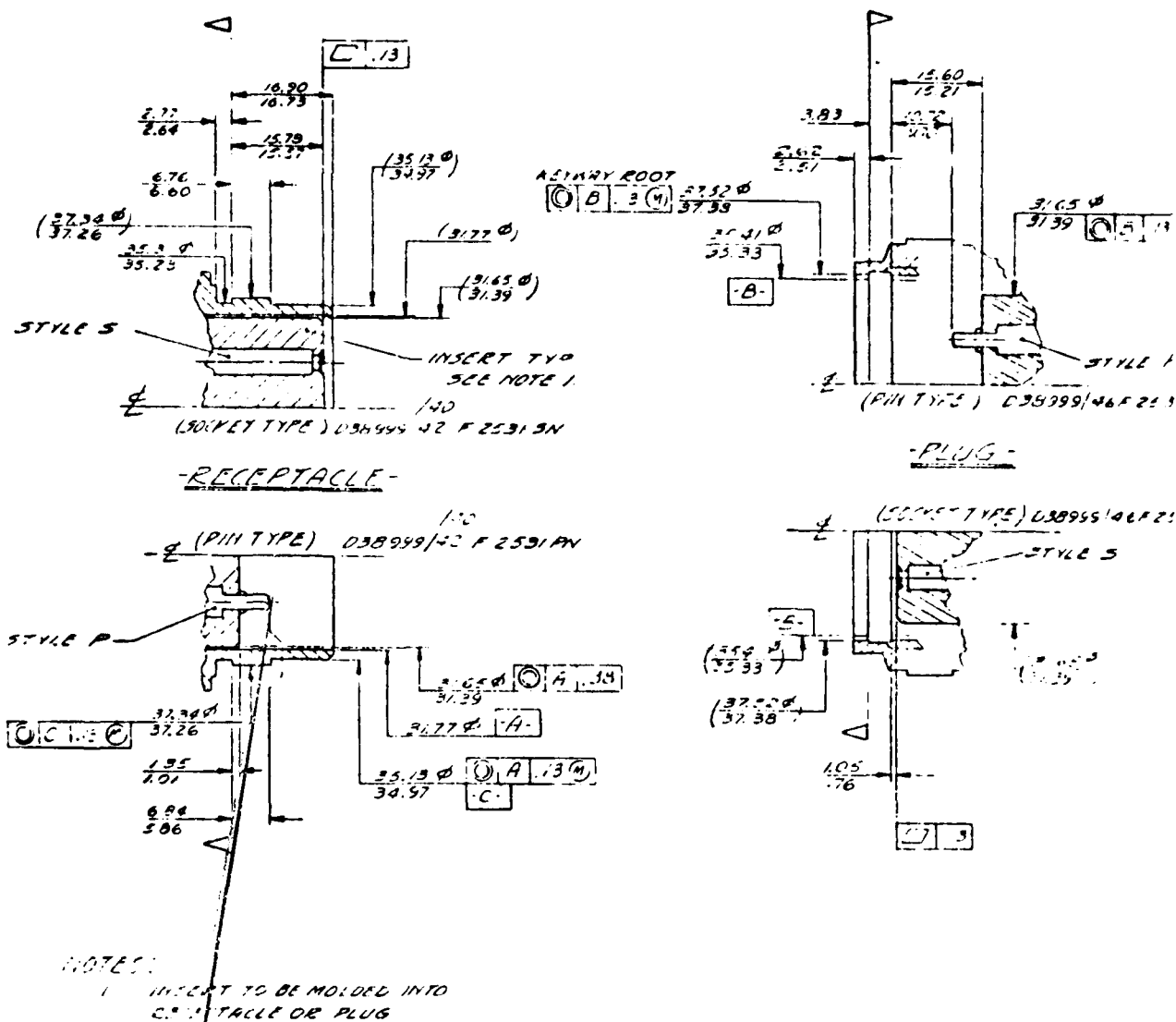


Figure 4. Receptacle - Plug, Insert Detail (Style S&P)

Ada Programming Language

Because of space constraints in this Proceedings, the Ada Programming Language Reference Manual dated July 1980 has not been reproduced. If you are interested in obtaining a copy, send your requests to:

Chairman,
DoD Management Steering Committee
for Embedded Computer Resources
Room 2A318
The Pentagon
Washington, D.C. 20301

