CR-152288

FEASIBILITY STUDY

FOR A

NUMERICAL AERODYNAMIC SIMULATION FACILITY

(NASA-CR-152288) FEASIBILITY STUDY FOR A NUMERICAL AERODYNAMIC SIMULATION FACILITY. VOLUME 2: HARDWARE SPECIFICATIONS/DESCRIPTIONS Final Report (Control Data Corp., St. Paul, Minn.) 474 p G3/09

N79-26069

Unclas 28382

Volume II – Hardware Specifications/Descriptions

Contributions by: F. M. Green
D. R. Resnick

MAY 1979

Distribution of this report is provided in the interest of information exchange. Responsibility for the contents resides in the authors or organization that prepared it.

Prepared under Contract No. NAS2-9896

CONTROL DATA CORPORATION
Research and Advanced Design Laboratory
4290 Fernwood Street
St. Paul, Minnesota 55112

for

AMES RESEARCH CENTER

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

FEASIBILITY STUDY
FOR A
NUMERICAL AERODYNAMIC SIMULATION FACILITY

Volume II — Hardware Specifications/Descriptions

Contributions by: F. M. Green
D. R. Resnick

Distribution of this report is provided in the interest of information exchange. Responsibility for the contents resides in the authors or organization that prepared it.

Prepared under Contract No. NAS2-9896

CONTROL DATA CORPORATION
Research and Advanced Design Laboratory
4290 Fernwood Street
St. Paul, Minnesota 55112

for

AMES RESEARCH CENTER

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

TABLE OF CONTENTS

Page

DIVISION 1 FMP FUNCTIONAL COMPUTER SPECIFICATION
(Detailed division table of contents begins on page ii of the division.)

DIVISION 2 FMP INSTRUCTION SPECIFICATION
(Detailed division table of contents beings on page ii of the division.)

DIVISION 3 STANDARD PRODUCT SYSTEM COMPONENTS

CYBER 170/Model 175-100 Computer System 3-1
7030 CYBER Extended Core Storage. 3-10
255X Network Processing Unit. 3-12
10315 Data Channel Converter. 3-18
3270/8271 Transfer Switch Subsystem 3-19
405 Card Reader 3-20
3447 Card Reader Controller 3-21
415 Card Punch. 3-23
3446 Card Punch Controller. 3-24
580 Train Printer 3-26
677 Magnetic Tape Transport 3-29
679 Magnetic Tape Transport 3-30
7021-3X Magnetic Tape Controller. 3-32
885 Fixed Module Drive. 3-34
7155 Fixed Module Drive Controller. 3-36
Mass Storage Subsystem. 3-38
819 Disk Storage Unit 3-41
7639 Mass Storage Controller. 3-42
CYBER 18/Model 20 3-44
1882 MOS Main Memory Storage. 3-48
1811-2 Operator Console 3-49
1843-1 Dual-Channel Communication Line Adapter. 3-50
1828-1 Card Reader/Line Printer Controller. 3-52
1829-60 Card Reader 3-53
1827-60 Line Printer. 3-54
1833-1 Storage Module Drive Interface 3-56
1833-3 Storage Module Control Unit. 3-57
1867-20/21 Storage Module Drive 3-58
1860-1/2/3/4 Magnetic Tape Subsystems 3-59

DIVISION 4 LOOSELY COUPLED NETWORK SPECIFICATION/DESCRIPTION

APPENDIX A - PERFORMANCE OF TRUNK ALLOCATION CONTENTION ELIMINATION (TRACE) METHOD. 4-A-1

APPENDIX B - LCN CHANNEL PROTOCOL 4-B-1

B1.0 INTRODUCTION 4-B-1
B2.0 MESSAGE FRAME STRUCTURE. 4-B-2
B2.1 General. 4-B-2
B2.1.1 Command Message Frame Structure. 4-B-2
B2.1.2 Response Message Frame Structure 4-B-5
B2.2 Frame Elements 4-B-5
B2.2.1 Frame Synchronization (P, F) 4-B-5
B2.2.2 Destination Address Field (T). 4-B-7

TABLE OF CONTENTS

		Page
B2.2.3	Function Field (FUN)	4-B-7
B2.2.3.1	Command Function	4-B-7
B2.2.3.2	Response Function.	4-B-8
B2.2.4	Access Code Field (A1,A2).	4-B-8
B2.2.5	Resync Parameter Field (RP).	4-B-8
B2.2.6	Source Address Field (S)	4-B-9
B2.2.7	Length Field (L1,L2)	4-B-9
B2.2.8	Header Frame Check Sequence Field (FC1,FC2).	4-B-9
B2.2.9	Information Field (I).	4-B-10
B2.2.10	Information Frame Check Sequence Field (FC3,FC4).	4-B-11
B2.2.11	Parameter Fields (P1,P2,P3).	4-B-11
B2.3	Additional Conventions	4-B-11
B2.3.1	Intermessage Time Fill	4-B-11
B2.3.2	Abort.	4-B-11
B2.3.3	Invalid Message.	4-B-12
B2.3.4	Order of Bit Transmission.	4-B-12
B2.3.5	Compatibility.	4-B-12
B3.0	DESIGN GUIDELINES.	4-B-12

APPENDIX C - PROPOSED LCN UNIFIED SECOND LEVEL PROTOCOL 4-C-1

DIVISION 1

FMP FUNCTIONAL COMPUTER

SPECIFICATION

CDC(R) FLOW MODEL PROCESSOR

Functional Computer

Specification

TABLE OF CONTENTS

		Page
1.0	SCOPE	1
2.0	APPLICABLE DOCUMENTS	2
3.0	REQUIREMENTS	3
3.1	General Functional Description	3
3.2	Scalar Unit	12
3.2.1	Scalar Unit Error Checking	15
3.2.2	Associative Unit	16
3.2.3	Instruction Issue/Decode	16
3.2.4	Register File	18
3.2.5	Branch/Instruction Stack	18
3.2.6	Main Load/Store	19
3.2.7	Intermediate Load/Store	19
3.2.8	Scalar Floating Point	20
3.2.9	Trace Register	28
3.2.10	Exchange Operation and Interrupts	29
3.3	Streaming Control Unit	35
3.3.1	Instruction Entry to Streaming Control	37
3.3.2	Dependency and Interlock Flags	38
3.3.2.1	Interlock Flags	38
3.3.2.2	Dependency Flags	39
3.3.3	Execution Queues	40
3.4	Vector Processor	42
3.4.1	Vector Ensemble (VE)	45
3.4.1.1	Read Bus Select Elements	47
3.4.1.2	Write Bus Select Elements	49
3.4.1.3	Front-End Add Elements	49
3.4.1.4	Multiply Elements	51
3.4.1.5	Back-end Adder Elements	51
3.4.1.6	Divide Table Element	52
3.4.1.7	Error Checking	56
3.4.1.7.1	SECDED	56
3.4.1.7.2	Parity	56
3.4.1.7.3	Result Checking	57
3.4.1.8	32/64-Bit Arithmetic	58
3.4.1.9	Asynchronous Control	60
3.4.1.10	Control Signals	60
3.4.1.11	Microcode Terms	60
3.4.1.12	Interface Timing	60
3.4.2	Vector Streaming Unit	61
3.4.2.1	Read Ports of VSU	64
3.4.2.1.1	Port Memory Addressing	64
3.4.2.1.2	Port Data Management	67
3.4.2.2	FIFO Function	68
3.4.2.3	Pipeline Selection	71

TABLE OF CONTENTS

		Page
3.4.2.3.1	Normal Operation of Selection Networks. . . .	73
3.4.2.3.2	Error Recovery and Maintenance.	74
3.4.2.4	Write Ports of VSU.	74
3.4.2.4.1	Addressing.	74
3.4.2.4.2	Data Management	75
3.5	Map Units	76
3.5.1	Main Map Unit	76
3.5.1.1	READ 1 and READ 2	78
3.5.1.1.1	Address Management.	78
3.5.1.1.2	Data Management	78
3.5.1.2	READ 3 Port	78
3.5.1.3	COMPRESS Element.	79
3.5.1.4	MASK/MERGE Element.	81
3.5.1.5	ASSEMBLY Element.	81
3.5.1.6	WRITE Port.	82
3.5.1.7	Operational Descriptions.	82
3.5.1.7.1	Gather.	82
3.5.1.7.2	Scatter	83
3.5.1.7.3	Compress.	83
3.5.1.7.4	Mask/Merge.	84
3.5.1.8	Operations with the Intermediate Map Unit . .	84
3.5.2	Intermediate Map Unit	85
3.6	Memory Interchange.	87
3.6.1	Ports	89
3.6.1.1	Read Ports.	89
3.6.1.2	Write Ports	90
3.6.2	Priority and Memory Control	90
3.7	Main Memory	92
3.7.1	Memory Module	93
3.7.2	Memory Configuration.	95
3.8	Intermediate Memory	97
3.8.1	Organization and Access	97
3.8.2	High Speed Ports.	97
3.8.3	Low Speed Ports	99
3.9	Input/Output.	99
3.9.1	I/O Unit.	99
3.9.1.1	The PDC	101
3.9.1.2	I/O Ports	102
3.9.2	Swap Unit	104
3.9.2.1	Scalar Interface.	106
3.9.2.2	Backing Store Interface	106
3.10	Backing Store	107
3.11	Maintenance Control Unit (MCU).	108
3.11.1	MCU/CPU Interface	108
3.11.1.1	Channels from CPU to MCU.	109
3.11.1.2	Channels from MCU to CPU.	113
3.11.2	MCU/Microcode Memory Interface.	125

TABLE OF CONTENTS

		Page
3.11.2.1	Microcode Units and Addresses	125
3.11.2.2	Microcode Error Checking.	125
3.11.2.3	MCU Interface Channel Bits.	126
3.11.3	Microcode Memory Channel Programming.	126
3.11.3.1	Typical Microcode Interface Function Codes. .	126
3.11.3.2	Microcode Switches.	128
3.11.3.3	Stream Microcode Status	131
3.11.3.4	Interface Sequences	132
3.11.3.5	Writing or Sweeping Scalar Microcode Memories.	135
3.11.3.5.1	Scalar Microcode Memory Write Operations. . .	135
3.11.3.5.2	Scalar Microcode Memory Sweep Operations. . .	136
3.11.4	Monitoring System Activity by the MCU	137
3.11.4.1	Monitoring with Counters.	137
3.11.4.1.1	MCU Count Gates and CPU Lines	142
3.11.4.1.2	Carry Lines	143
3.11.4.1.3	Stop Lines.	143
3.11.4.1.4	Counter Setup	143
3.11.4.2	Display Registers	144
3.11.4.3	Logic Fault Monitoring.	148
3.11.5	SECDED (Single Error Correction Double Error Detection).	148
3.11.6	Absolute Bounds Address	151
3.12	Timing Information.	151
3.12.1	Scalar Unit Timing.	151
3.12.1.1	Use of Scalar Timing Tables	152
3.12.1.2	Basic Instruction Timing.	159
3.12.2	Vector Processor Timing	167
3.12.3	Map Unit Timing	172
3.12.4	Swap Unit Timing.	173
4.0	QUALITY ASSURANCE PROVISIONS.	174
5.0	PREPARATION FOR DELIVERY.	174
6.0	NOTES	174
6.1	Intercom.	174
6.2	System Startup.	174
APPENDIX A -	Unique Syndrome Words for Single Bit Failures.	175
APPENDIX B -	Asynchronous Data Movement Control For the Flow Model Processor.	179

TABLE OF CONTENTS

		Page
APPENDIX D -	FMP Functional and Timing Characteristics	186
D1.0	Introduction	186
D2.0	Instruction Issue	187
D2.1	Issue Timing Parameters	188
D3.0	Main Memory	189
D3.1	Main Memory Access	190
D3.2	Main Memory Access Timing Parameters	191
D4.0	Streaming Control	191
D4.1	Interlock Flags	193
D4.2	Keys and the Data Flag Branch Registers	194
D4.3	Streaming Control Unit Timing Parameters	194
D5.0	Vector Operations	194
D5.1	Vector Read Port	195
D5.2	FIFO Buffer	196
D5.3	Pipelines	196
D5.4	Vector Write Port	197
D5.5	Vector Operation Timing Parameters	197
D6.0	Map Units	198
D6.1	Main Map Unit	198
D6.2	Intermediate Map Unit	200
D6.3	Combined Map Unit Operations	201
D6.4	Map Unit Timing Parameters	201
D7.0	Intermediate Memory	201
D7.1	Organization and Access	202
D7.2	High Speed Ports	202
D7.3	Low Speed Ports	202
D8.0	Input/Output	203
D8.1	I/O Channels	203
D8.2	Swap Unit	205
D8.3	Scalar Unit Access to Intermediate Memory	205
APPENDIX E -	Checking Functions During Pipeline Processing	207

1.0 SCOPE

The CDC Flow Model Processor (FMP) Functional Specification is intended to provide information of the following types to either the user or maintenance personnel.

- Information that may be obtained about computer/program operation via the Maintenance Control Unit (MCU).
- Changes in mode or operation internal to the FMP that may be made via the MCU or program that are not specified in the FMP Instruction Specification.
- Information concerning computer operation that is of value in debugging software/hardware or in program optimization.

This specification is not intended to provide information as to how a unit performs its specified tasks such as would normally be found in a Theory of Operation.

2.0 APPLICABLE DOCUMENTS

10354636 CDC Flow Model Processor Instruction Specification

3.0 REQUIREMENTS

3.1 General Functional Description

The Flow Model Processor (FMP) is an extremely high speed computational system designed to provide solutions to very large systems of differential equations with particular emphasis on solutions of problems relating to aerodynamic flow. The processing system is based, in part, on the Control Data STAR-100 architecture, with both Main Memory and Scalar Unit design being taken from the STAR-100 family (for descriptions of these units see section 3.2 and 3.7).

The resulting basic structure is augmented by the addition of several very high performance memories and computational elements:

1) A Backing Store Unit that can hold from 32 million to 128 million 64-bit words. The memory is nominally 128 million words. The memory has full SECDED protection and is designed with future expandability in mind. See section 3.10.

2) An Intermediate Memory Unit that can hold 32 million 64-bit words. This is a higher performance memory than the Backing Store (and lower performance than Main Memory). This memory has full SECDED and also has the capability for future expansion. See section 3.8.

3) A Swap Unit that moves data between the Backing Store and Intermediate Memory. This unit also contains an access port to Intermediate memory for use by the Scalar Unit for scalar access to the Intermediate Memory. See section 3.9.

4) Two Map Units that can, singly or working together, reorder (map) and move data in Intermediate Memory, Main Memory, or between the two memories. See section 3.5.

5) A very high performance set of Vector Units (pipelines) that perform the major computational work of the FMP. See section 3.4.

6) A Vector Streaming Unit that controls and manages memory addressing and data control for the vector pipelines. See section 3.4.

7) A Streaming Control Unit that provides control and buffering of instructions to be executed by the Map and Vector Units. See section 3.3.

(continued)

3.1 (Cont.)

A block diagram of this resulting architecture is shown in figure 3.1-1. For future expandability in the event that technology proceeds faster than expected, lines are provided for two additional address bits on all address paths to the memories. These bits will allow expansion of each memory to four times its stated maximum size, if and when technology permits it: Main Memory to 32 million words; Intermediate Memory to 128 million words; Backing Store to 512 million words.

Data and programs are entered into the FMP via the Input/Output ports attached to the Intermediate Memory. Normally the data and programs will be moved to the Backing Store by the Swap Unit as the data is entered into the Intermediate Memory. Once the various fragments of a job are aggregated in the Backing Store, and the FMP is idle, the job is "rolled" into the Intermediate Memory via the Swap Unit. A portion of the job--possibly the whole job if it will fit Main Memory--is then moved into Main Memory. Certain portions of the computations, all of the bookkeeping and all of the FMP's overall control are accomplished in the Scalar Processor . It is this processor that interprets the instruction stream, acts on those instructions which it can, and distributes the remaining instructions to the appropriate attached units (Vector, Main Map, Intermediate Map). Commands to the Swap and I/O Units are put in command buffers in the Intermediate memory. The Swap and I/O Units interpret their respective buffers upon receipt of a signal from the Scalar Processor.

The FMP is designed to operate at a minor clock cycle rate of 16 nanoseconds, with all data transfers, and all pipeline segments capable of clocking a new data quantity (32, 64, 128, 512, or 1024 bits wide) every minor cycle. The maximum rate of arithmetic results produced in the 4 vector pipelines then becomes 3(operational peak rate)*2 (32-bit results per pipleline)*4(pipelines) *2(results per clock period) = 48 results per minor cycle of 16 nanoseconds=3.0 billion floating-point operations per second.

The Scalar, Map, Swap, and Vector Units are capable of operating simultaneously so that a majority of bookkeeping and data mapping (reorganization functions) can be overlapped with the computation. This enables the effective rate of problem solution to approach 60% of the peak rate, or 1.8 billion operations per second, which exceeds the original objectives established for Navier-Stokes solutions for flow field simulations.

(continued)

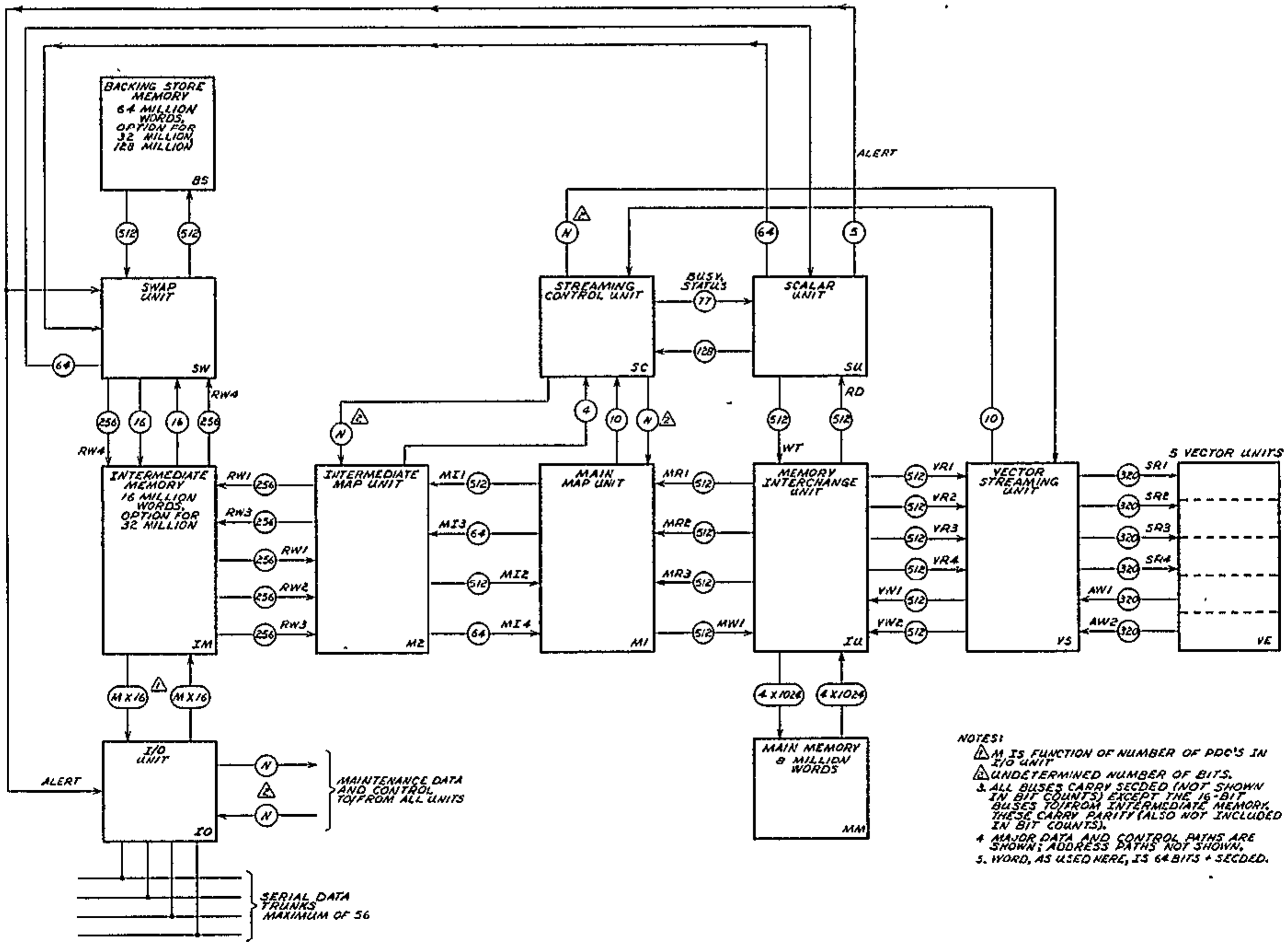


Figure 3.1-1 Basic CDC FMP Configuration

3.1 (Cont.)

Unlike the STAR-100, the FMP is designed for monoprogramming of computational jobs, thus there is no virtual memory mechanism. All user jobs are given the use of the entire eight million words of Main Memory minus the first 65K words which are reserved for the FMP monitor. This monitor area cannot be accessed by the job mode programs. A series of several monitor mode instructions permits the management and allocation of the Intermediate Memory and Backing Store, as well as control communications with the I/O processors attached to the FMP. These I/O processors, called PDCs (Programmable Device Controllers), are capable of intelligent control of the I/O trunks (up to four attached to each PDC) and intelligent communications with the monitor, as well as providing 200-megabit data transfer rates between the Intermediate Memory and the trunks, and 50-megabit transfer rate on the actual coax trunks themselves. (200-Mbit/sec rate is for four PDCs performing I/O. The I/O system is designed to allow up to 14 PDCs all moving data simultaneously.

The instruction set for scalar operations is a compatible subset of the STAR-100 family which supports most STAR software, with the addition of a few operations made necessary by the unique I/O and Backing Store configuration provided on the FMP. The Map Units provide execution capability for the STAR-100 "Iverson" operators of vector MASK,MERGE,COMPRESS and SCATTER/GATHER while the Vector Unit, in cooperation with the Map Unit, performs the "Iverson" SELECT, SEARCH, and SEARCH INDEXED LIST operations.

The Vector Unit, in conjunction with the Vector Streaming Unit, performs the add, subtract, multiply, divide operations commonly found on most processors, in addition to a series of linked and macro operations providing combinations of additions and multiplications every minor cycle. The set of linked operations chosen were based on the characteristics of flow-model simulations that have been analyzed by Control Data Corporation. In addition to the simple combinations of add/subtract and multiply, the functions SUM,PRODUCT,SUM OF PRODUCTS, and PRODUCT OF SUMS are included for matrix computations. The Vector Unit, with the Vector Streaming Unit, can also form bit strings (called control vectors) that can be used by the Map Units.

To ensure the reliability and maintainability of the FMP, a number of error checking and recovery facilities are built in, as well as a group of maintenance functions which can be invoked by a designated computer attached to one or more of the

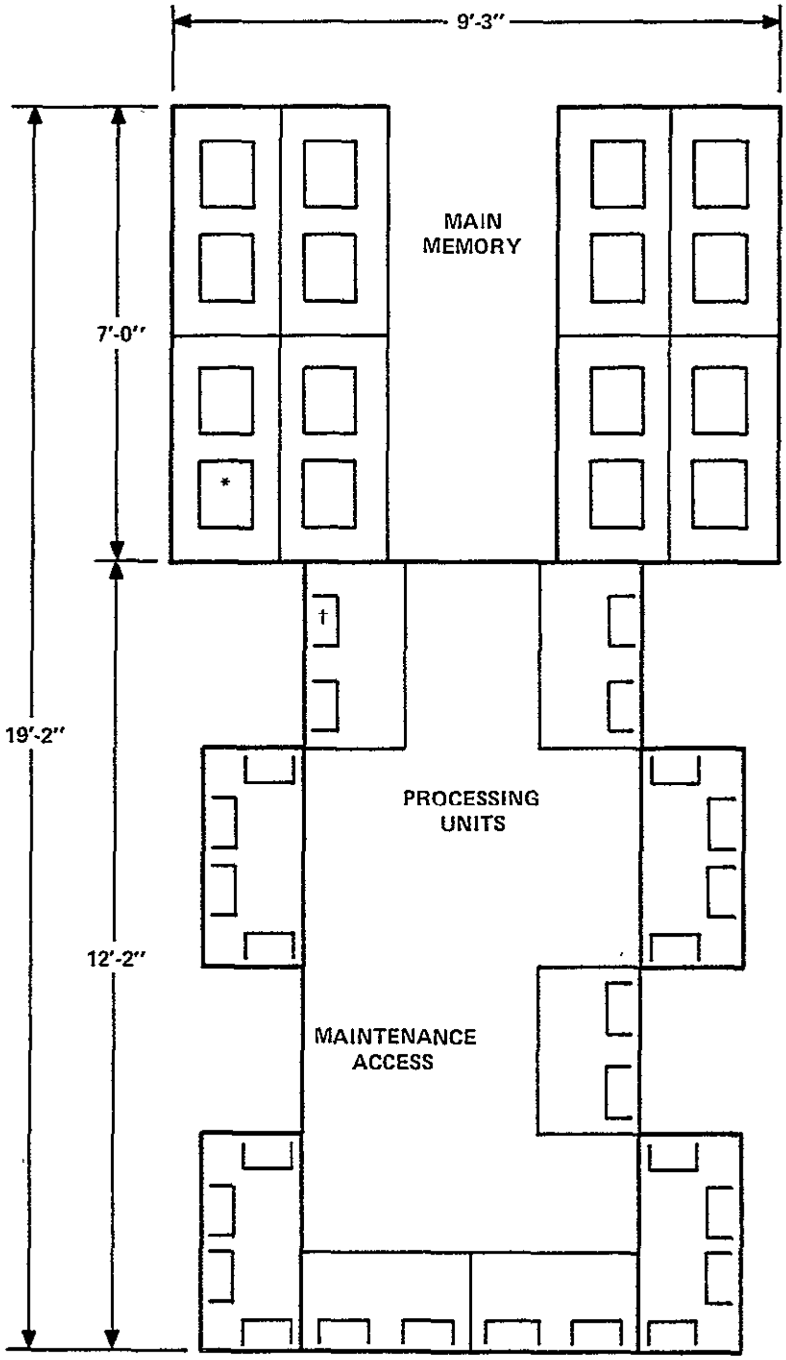
(continued)

3.1 (Cont.)

I/O trunks. Single Error Correction, Double Error Detection (SECDED) is carried through all data trunks up to the functional unit actually using the data. Checking for errors is done at several points in the data path (for example at the Memory, at the Vector Streaming Unit, and at the Vector Unit) so that faults can be quickly isolated, while the error correction is applied at the point where the data is used, for example the input stream of the Vector Unit.

The physical layout with dimensions of the FMP is shown in figures 3.1-2 and 3.1-3. The configuration of LSI panels within the two basic types of cabinets is shown in figures 3.1-4 and 3.1-5.

(continued)



NOTES:

* COLUMN OF FOUR MEMORY CHASSIS

† COLUMN OF FOUR LSI CHASSIS

Figure 3.1-2 CDC FMP Floor Layout (Processor and Main Memory)

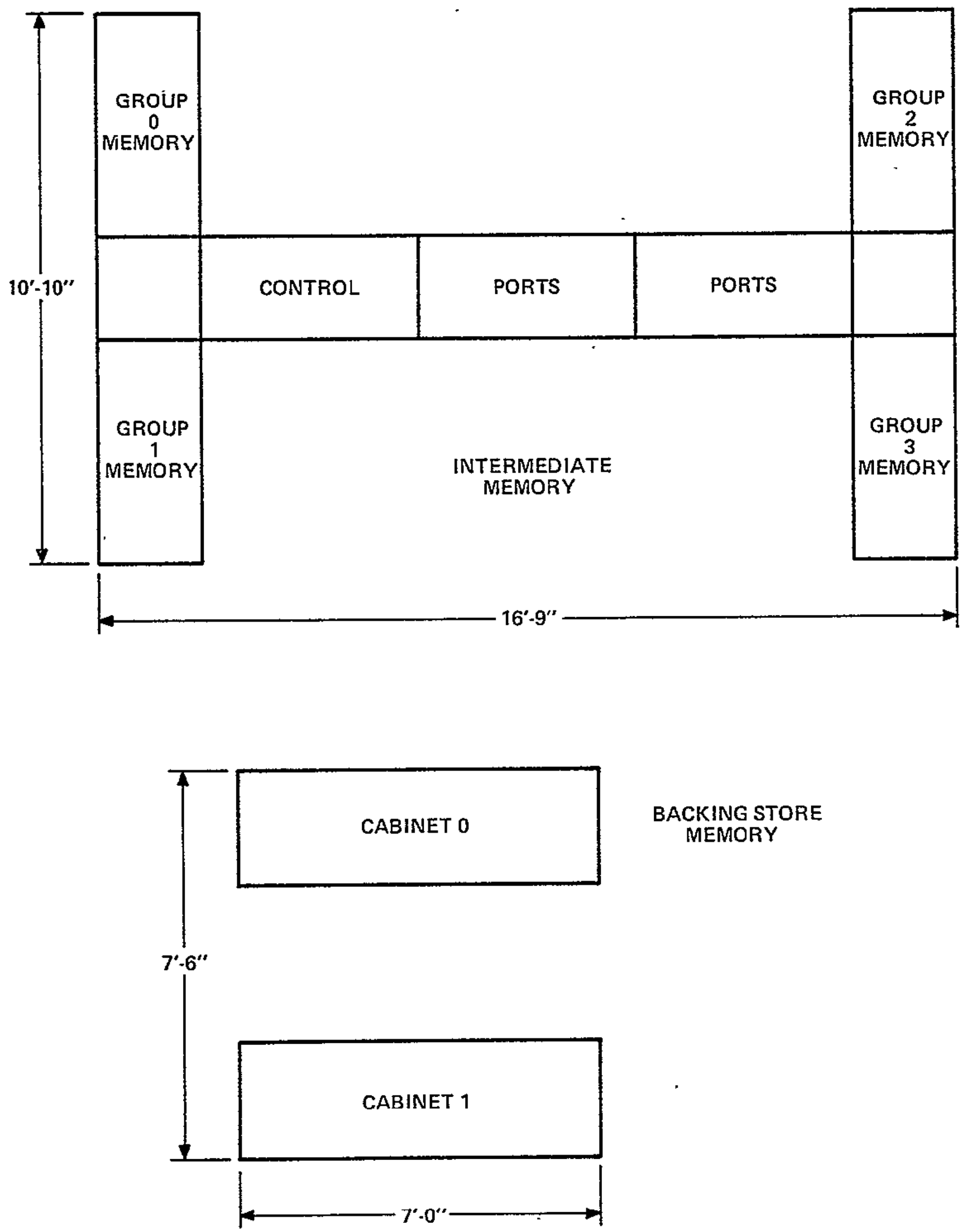


Figure 3.1-3 CDC FMP Floor Layout (Intermediate Memory and Backing Store)

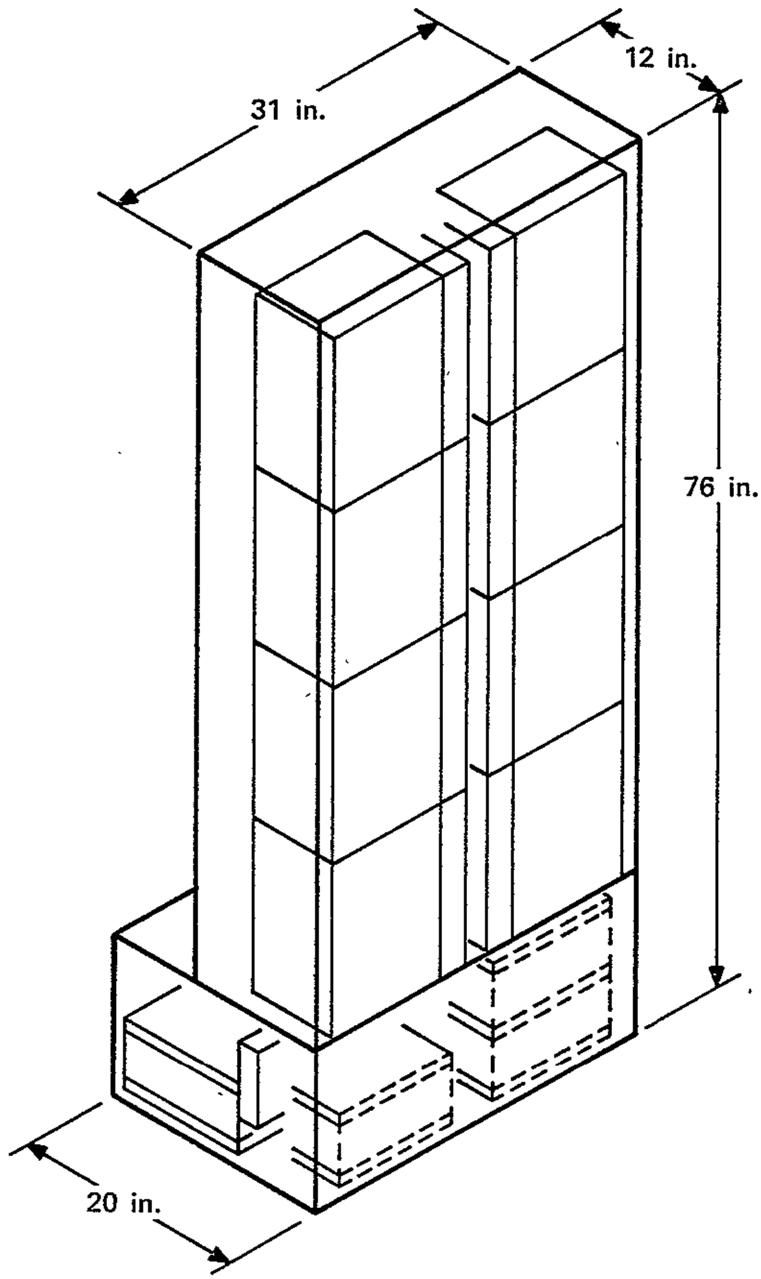


Figure 3.1-4 Cabinet for 8 LSI Panels

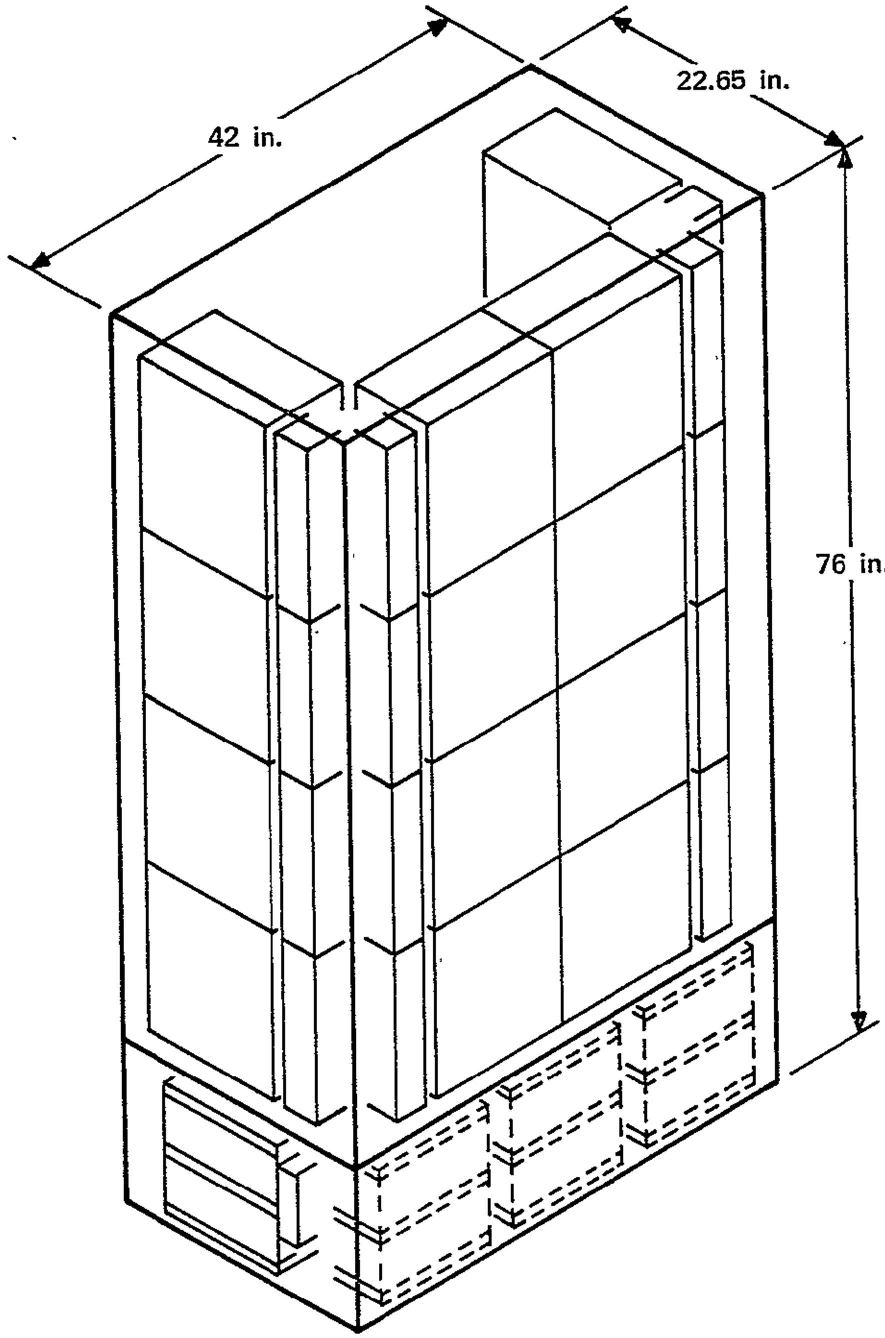


Figure 3.1-5 Cabinet for 16 LSI Panels

3.2 Scalar Unit

The Scalar Unit is physically contained in a cabinet attached to the Vector Processor cabinet. Main Memory is attached to the Scalar Unit cabinet in order to reduce transfer delays and to gain performance.

The Scalar Unit has synchronous internal logic with a clock period of 16 nanoseconds and is implemented using LSI circuits. A block diagram of the functional components of the Scalar Unit is shown in Figure 3.2-1.

The CDC FMP instruction control is contained in the Scalar Unit. The Instruction Issue Unit receives and decodes all instructions from Main Memory. A semiconductor instruction stack provides buffering for eight swords of instructions (512 bits per sword). The stack can contain up to 128 32-bit instructions or 64 64-bit instructions or a mixture. The instruction stack can contain up to 6 discontiguous swords with two sword lookahead. The Read Next Sword (RNS) portion of RNS/Branch provides the control for loading the instruction stack. The Branch portion performs branch condition testing and executes the branch instructions.

Instruction Issue is pipelined and is capable of issuing instructions at the rate of one instruction every 16 nanoseconds. The Instruction Issue Unit decodes all instructions and directs decoded stream instructions to the appropriate unit for execution. Thus, with independent vector and scalar instruction controls operating on a single instruction stream, the Scalar Unit can execute scalar instructions in parallel with stream instructions.

If an instruction is referenced which is not presently in the stack, Instruction Issue is halted and a memory request is made for the word containing the required instruction. The sword thence brought from memory must replace one of the swords already in the stack. The sword that is "thrown away" or overlaid by the incoming sword is the least recently used (LRU) sword. Thus if swords numbered consecutively 0 through 7 have been executed without any intervening branches, sword 8

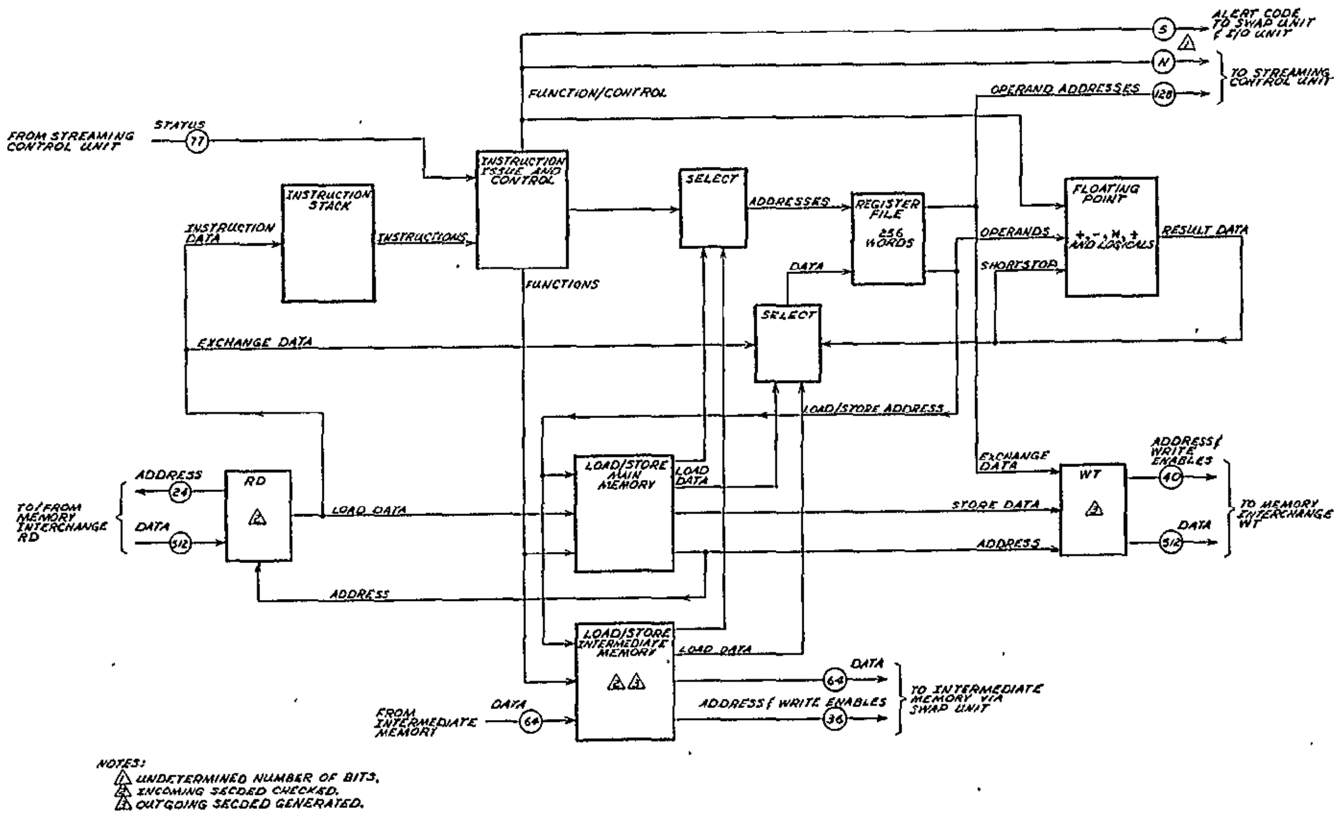


Figure 3.2-1 Scalar Unit Block Diagram

3.2 (Cont.)

(required by the next consecutive instruction) would be brought from memory and overlaid in the stack in the position originally held by sword number 0 which, in this case, is the LRU sword.

Main Load/Store provides special handling of the Main Memory Load and Store instructions. The unit acts as a pipeline and is capable of accepting a new request rate of one load every minor cycle or one store every two minor cycles, provided a memory busy or register file write-bus busy does not occur. A circular buffer containing six registers provides buffering for up to six load requests, or three store requests, or a mixture of loads and stores.

Main Load/Store is capable of loading a randomly accessed word of data from Main Memory into the Register File 15 cycles after reading the base address and item count of the data from the Register File. This time assumes a memory busy or register file write-busy bus does not occur. A memory busy would add up to 3 cycles to the load time.

The Intermediate Load/Store provides special handling of scalar requests to Intermediate Memory just as the Main Load/Store does for requests to Main Memory. The Intermediate Load/Store has the same features as Main Load/Store - pipelined execution using a circular buffer. Because Intermediate Memory has a slower access time than Main Memory, the load time for a load request will take about 23 clock cycles instead of 15 as in the Main Load/Store. A memory busy could add up to 24 clock cycles to the load time.

Scalar Floating Point contains completely independent functional elements to attain high scalar performance. The following are the times in clock cycles to produce a 32-bit or 64-bit result in each functional element. These times correspond to the shortstop times. Shortstop is the process by which a result from any arithmetic element may be returned directly to either input of any arithmetic element. This occurs in parallel with the storing of the result in the Register File. Shortstop eliminates the time necessary to store the result in the

(continued)

3.2 (Cont.)

Register File and then retrieve it for use in the next arithmetic operation.

Add/Subtract Pipe 5 cycles Multiply Pipe 5 Shift/Logical Pipe 4 Single Cycle Pipe 1 Divide/SQRT/Convert Element 24

The pipe elements are segmented and capable of accepting new operands every clock cycle. The Divide/SQRT/Convert element must complete each operation before a new one can begin. All elements are capable of being shortstopped. The Scalar Unit contains a semiconductor Register File which provides 256 64-bit registers for use in instruction and operand addressing, indexing, field lengths, and as source and destination registers for scalar instruction operands and results. The Register File is capable of two reads and one write every clock cycle.

3.2.1 Scalar Unit Error Checking

The basic design of the FMP Scalar Unit is based on the design of the STAR-100A and STAR-100B Scalar Units. In these designs (already being implemented) there exists a moderate amount of error checking on buses:

a. SECDED - All data buses in and out of the Scalar Unit carry seven bits of single error correction, double error detection code bits for each 32 bits of data. The data buses are the Load/Store data bus, the Instruction Read data bus, the Register File exchange path, and the Register File data bus to the Streaming Control Unit. See section 3.11.5 for additional information on SECDED.

b. Parity - All microcode memories in Instruction Issue and Scalar Floating Point contain parity bit checking. The microcode carries a parity bit from the time it is assembled on a front-end processor, until it is read during execution in a given unit. A parity fault causes an immediate stoppage of the CPU, and an error flag to be sent to the Maintenance Control Unit (MCU). The instruction stack contains parity information in like manner.

(continued)

3.2.1 (Cont.)

c. Illogical function - Communication between the various functional elements of the Scalar Unit is performed by sequences of microcode generated function codes, which are decoded at the receiving end by microcode. Sufficient entropy has been included in the function code scheme to permit some detection of internal control signal failures.

3.2.2 Associative Unit

N/A

3.2.3 Instruction Issue/Decode

All instructions are read from memory by the Scalar Unit and decoded for subsequent issue. This is accomplished in Instruction Issue. After an instruction is decoded it is issued to the unit responsible for further action: the Streaming Control Unit or the Scalar Unit itself. The Scalar Unit proceeds with execution of instructions issued to it. The Streaming Control Unit receives instructions to be executed by the Vector or Map Units; after checking keys and setting appropriate flags, the Streaming Control Unit places these instructions in their respective queues. Responsibilities for all instructions are shown in table 3.2-2.

These units are essentially independent of one another and each can execute instructions in parallel. The remainder of section 3.2 provides additional information on Scalar Unit operation. Section 3.3 describes the Vector Unit and section 3.9 covers the Swap Unit.

(continued)

TABLE 3.2-2 INSTRUCTION RESPONSIBILITY

Rows: Second Digit of Instruction Code; Columns: First Digit of Instruction Code

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	S	S	S	S	S	S	S	S	I	I	I	S	I	I	I	I
1	I	S	S	S	S	S	S	S	I	I	I	S	I	I	I	I
2	I	S	I	S	S	S	S	S	I	I	I	S	I	I	I	I
3	I	S	I	S	I	S	S	S	I	I	I	S	I	I	I	I
4	S	I	S	S	S	S	S	S	I	I	I	S	I	I	I	I
5	I	I	S	S	S	S	S	S	I	I	I	S	I	I	I	I
6	S	I	S	S	S	I	S	S	I	I	I	S	I	I	I	I
7	I	I	S	S	I	I	S	S	I	I	I	I	I	I	I	I
8	S	I	I	S	S	S	S	S	I	I	I	I	I	I	I	I
9	S	I	I	S	S	S	S	S	I	I	I	I	I	I	I	I
A	S	I	I	S	I	S	I	S	I	I	I	I	I	I	I	I
B	I	I	S	S	S	S	S	S	I	I	I	I	I	I	I	I
C	I	I	S	S	S	S	S	S	I	I	I	I	I	I	I	I
D	I	I	S	S	S	S	S	S	I	M	I	I	S	I	I	I
E	S	I	S	S	S	S	S	S	I	V	I	S	S	I	I	I
F	I	I	S	S	S	S	S	S	I	V	I	S	I	I	I	I

S - Executed within the Scalar Processor (Note that Data Flag information will be passed to the Data Flag Register for appropriate instructions).

V - Executed in the Vector Processor (Vector Streaming Unit and Vector Unit).

M - Executed in the Map Unit.

I - Illegal instruction.

3.2.4 Register File

The Register File of the FMP contains 256 64-bit words. This Register File is capable of accomplishing two read operations and one write operation every minor cycle. In addition, the Register File can be exchanged at the rate of two registers in and two out every minor cycle. A complete swap of the Register File is accomplished in 128 minor cycles plus set-up time.

The FMP has 16 Result Address Registers (RAR) used to conflict check each instruction ready for issue against register file addresses that are to be written by a previously issued scalar instruction. If a conflict exists, the action taken depends on whether the needed result can be shortstopped or not (see sections 3.2 and 3.2.8 for additional information on shortstop). If shortstop is possible, the instruction is issued at the appropriate time and instruction issue continues. If shortstop is not possible (e.g., the result of a previous load from memory instruction is needed), issue stops.

The RARs are set sequentially from the result register designators of issued scalar instructions. They are cleared when the result is written into the Register File.

3.2.5 Branch/Instruction Stack

The Branch instruction execution times may be found in section 3.12 of this Specification.

The instruction stack implemented in the FMP accommodates up to 8 swords (512 bits per sword), 6 of which may be discontiguous. To sustain the instruction rate a two-sword "lookahead" will be done by reading the two swords following the one being executed. Issue will not be blocked if the swords following are not in the stack.

An address is maintained for each of the eight swords so that out-of-stack branches may be taken without voiding the entire stack. For instance, it would be possible to call a subroutine of up to 3 swords (48 instructions/32 bits each) several times from a three sword instruction stream and never branch out-of stack after the first branch which loads the subroutine into the stack.

(continued)

3.2.5 (Cont.)

Refer to paragraph A2.0 of Engr. Spec 10354636 for information about restrictions on self-modifying programs.

3.2.6 Main Load/Store

Main Load/Store executes the 12, 13, 32, 5E, 5F, 7E, and 7F instructions. There are six address registers in Main Load/Store which enable requests to be stacked and executed in the proper order. The 12, 5E, and 7E instructions each require one register and can be executed (in the absence of memory conflicts) at the rate of one load per minor cycle. The 5F and 7F instructions each require two address registers and can be executed at one store per two minor cycles. The 13 and 32 instructions each require two address registers which are then busy for 17 minor cycles.

Main Load/Store is thus capable of streaming load/ store instructions (other than the 13 and 32) at one minor cycle per load and two minor cycles per store assuming no Memory or Register File conflicts. For example, a stream of N loads will execute in N + 14 minor cycles from the issue of the first load until the operand from the last load available in the Register File. A stream of N stores will execute in $2N + \left|\frac{N}{3}\right.$ minor cycles from issue of the first store until issue of the last store.

3.2.7 Intermediate Load/Store

Intermediate Load/Store executes the 24, 25, 26, and 27 instructions. There are six address registers which allow requests to Intermediate Memory to be stacked just as are requests to Main Memory. The timing of instructions going to Intermediate Load/ Store is the same as for Main Load/Store--a load instruction takes one cycle to issue and a store instruction takes two cycles to issue. Because the Intermediate Memory has lower performance than Main Memory, Intermediate Load/Store is unable to maintain a streaming data rate. In order to increase the effective data rate to Intermediate Memory, the Swap Unit (which is the interface to Intermediate Memory for the Scalar Unit) has two buffers of 32 64-bit words. If the data is in a buffer the data can be sent taken immediately from the buffers instead of making an intermediate memory transfer request.

3.2.8 Scalar Floating Point

The FMP has an arithmetic unit dedicated to scalar (non-vector) operations. This Scalar Floating Point is divided into five separate functional elements: one each for add/subtract, multiply and logical, a single cycle element for the add/subtract address and transmit type instructions, and one combining divide, square root and convert.

All elements of the scalar arithmetic unit are separately and independently controlled to allow concurrent operation. However, only one operand pair is issued to the arithmetic unit each minor cycle so this becomes the limiting factor determining the result rate from concurrent operations.

There are four effectively segmented pipeline elements which accept a new pair of operands every minor cycle. They each produce a 64 or 32-bit result every minor cycle. The divide/square root/ convert element is not segmented and thus accepts operands only at completion of the previous operation, every 28 minor cycles per 64-bit operand. Using 32-bit operands would approximately double the result rate of the divide/square root/convert instructions.

(continued)

3.2.8 (Cont.)

Interface Between Scalar Floating Point and Scalar Control

Input Trunks

There are three input trunks to Scalar Floating Point. The characteristics of these trunks are outlined in the following description. All input operands are treated as 64 or 32-bit floating-point quantities, except as noted. If an indefinite or machine zero floating-point operand is received, its coefficient will be set to all zeros.

A Input Trunk

This trunk is 64 bits wide. It receives 64 data bits from register location R in the following format:

64-Bit Mode

	0 15	16 63
Information	exponent	coefficient

32-Bit Mode

	0 7	8 15	16 39	40 63
Information	expo-nent	expo-nent (copy of 00-07)	coefficient	zeros

All bits transferred on this trunk should be held on the trunk for a period of one cycle measured at the input to Scalar Floating Point.

B Input Trunk

The B trunk receives data from register location S and is identical to the A trunk.

(continued)

3.2.8 (Cont.)

Control Trunk

The control trunk carries the signals which control Scalar Floating Point. It is made up of the following signals:

Control Address

The control address bits are the bits that select the proper set of internal control signals for the floating-point instruction being executed. There is a unique code for each instruction as listed in Table 3.2-3. Using the input data to Scalar Floating Point as a reference, these control bits must arrive at the floating-point logic 1.5 cycles ahead of the data and be valid for one cycle.

Mode Controls

The mode controls are Mode 64 In, Mode 64 Out, G-bits and Divide. The Mode 64 and G-bit lines must lead the input data by 1.0 minor cycles and the Divide signal must lead by 1.5 minor cycles. These should remain up for one cycle.

Issue Controls

These controls are S-Shortstop, R-Shortstop, S-Clockgate, R-Clockgate, S-Shortstop Enable, R-Shortstop Enable, and Go. These controls all must be valid one cycle ahead of the data. The Shortstop Enable signals enable the setting or clearing of the Shortstop control flip-flops. The Shortstop signals set flags that cause data to be clocked into the floating-point input registers when these flags are a one. The Go signal tells the arithmetic unit to begin processing the operands that are in the input registers.

(continued)

3.2.8 (Cont.)

TABLE 3.2-3 INSTRUCTION CODES

INSTR	M64 IN	M64 OUT	CONTROL ADDRESS	G-BITS	DIV.	CYCLE TIME	BUSY TIME	A TRUNK	B TRUNK	OUTPUT CONTROL
10	1	1	01		1	20	17	0	R	DT.,DB,DFLG39
11	1	1	02		1	53	50	0	R	DT,DB
20	1	1	10		0		0	R	S	
21	1	1	11		0		0	R	S	
2A	1	1	18		0	3	0	I	R	
2B	1	1	19		0	3	0	I	R	
2C	1	1	1A		0	3	0	R	S	
2D	1	1	1B		0	3	0	R	S	
2E	1	1	1C		0	3	0	R	S	
2F	1	1	1D	G2,G3	0	1	0	O	T	
30	1	1	1E		0	3	0	R	S	
31	1	1	1F		0	1	0	R	+1	
34	1	1	20		0	3	0	R	S	
35	1	1	21		0	1	0	R	-1	
36	1	1	22		0	1	0	CIAR	+20	
38	1	1	23		0	1	0	R	T	
3C	0	0	24		0	5	0	R	S	
3D	1	1	25		0	5	0	R	S	
3E	1	1	26		0	1	0	R	I	
3F	1	1	27		0	1	0	R	I	
40	0	0	28		0	5	0	R	S	DFLG42,43,46
41	0	0	29		0	5	0	R	S	DFLG42,43,46
42	0	0	2A		0	5	0	R	S	DFLG42,43,46
44	0	0	2B		0	5	0	R	S	DFLG42,43,46
45	0	0	2C		0	5	0	R	S	DFLG42,43,46
46	0	0	2D		0	5	0	R	S	DFLG42,43,46
48	0	0	2E		0	5	0	R	S	DFLG42,43,46
49	0	0	2F		0	5	0	R	S	DFLG42,43,46
4B	0	0	30		0	5	0	R	S	DFLG42,43,46
4C	0	0	31		1	30	25	R	S	DFLG41,42,43,46
4D	0	0	32		0	1	0	I	O	
4E	0	0	33		0	1	0	R	I	
4F	0	0	34		1	30	25	R	S	DFLG41,42,43,46
50	0	0	35		0	5	0	O	R	DFLG46

(continued)

3.2.8 (Cont.)

TABLE 3.2-3 INSTRUCTION CODES (Cont.)

	IN	OUT	ADDRESS			TIME	TIME	TRUNK	INSTR TRUNK	M64 M64 CONTROL OUTPUT CONTROL
51	0	0	36		0	5	0	0	R	DFLG46
52	0	0	37		0	5	0	0	R	DFLG46
53	0	0	38		1	30	26	0	R	DFLG43,45,46
54	0	0	39		0	5	0	S	R	DFLG42,43,46
55	0	0	3A		0	5	0	S	R	DFLG42,46
58	0	0	3B		0	1	0	R	0	
59	0	0	3C		0	5	0	0	R	DFLG42,43,46
5A	0	0	3D		0	3	0	0	R	
5B	0	0	3E		0	3	0	R	S	
5C	0	0	3F		0	5	0	0	R	DFLG43,46
5D	0	1	40		0	5	0	0	R	DFLG43,46
60	1	1	41		0	5	0	R	S	DFLG42,43,46
61	1	1	42		0	5	0	R	S	DFLG42,43,46
62	1	1	43		0	5	0	R	S	DFLG42,43,46
63	1	1	44		0	1	0	R	S	
64	1	1	45		0	5	0	R	S	DFLG42,43,46
65	1	1	46		0	5	0	R	S	DFLG42,43,46
66	1	1	47		0	5	0	R	S	DFLG42,43,46
67	1	1	48		0	1	0	R	S	
68	1	1	49		0	5	0	R	S	DFLG42,43,46
69	1	1	4A		0	5	0	R	S	DFLG42,43,46
6B	1	1	4B		0	5	0	R	S	DFLG42,43,46
6C	1	1	4C		1	54	49	R	S	DFLG41,42,43,56
6D(1)	1	1	4D		0	4	0	R	S	
6D(2)	1	1	4E		0	3	0	T	0	
6E	1	1	4F		0	3	0	R	S	
6F	1	1	50		1	54	49	R	S	DFLG41,42,43,46
70	1	1	51		0	5	0	0	R	DFLG46
71	1	1	52		0	5	0	0	R	DFLG46
72	1	1	53		0	5	0	0	R	DFLG46
73	1	1	54		1	54	50	0	R	DFLG43,45,46
74	1	1	55		0	5	0	S	R	DFLG42,43,46
75	1	1	56		0	5	0	S	R	DFLG42,46
76	1	1	57		0	5	0	0	R	DFLG42,43,46
77	1	0	58		0	5	0	0	R	DFLG42,43,46
78	1	0	59		0	1	0	R	0	
79	1	1	5A		0	5	0	0	R	DFLG42,43,46
7A	1	1	5B		0	3	0	R	0	
7B	1	1	5C		0	3	0	R	S	
7C	1	1	5D		0	3	0	R	0	

Note: The 6D instruction requires three references to the Register File; this takes two minor cycles. The "(1)" is the first and the "(2)" is the second.

(continued)

3.2.8 (Cont.)

TABLE 3.2-3 INSTRUCTION CODES (Cont.)

INSTR	M64 IN	M64 OUT	CONTROL ADDRESS	G-BITS	DIV.	CYCLE TIME	BUSY TIME	A TRUNK	B TRUNK	OUTPUT CONTR(
B0,G1=0	1	1	60	G1,2,3,4	0	3	0	A	X	
B0,G1=1	1	1	70	G1,2,3,4	0	5	0	A	X	DFLG46
B1,G1=0	1	1	61	G1,2,3,4	0	3	0	A	X	
B1,G1=1	1	1	71	G1,2,3,4	0	5	0	A	X	DFLG46
B2,G1=0	1	1	62	G1,2,3,4	0	3	0	A	X	
B2,G1=1	1	1	72	G1,2,3,4	0	5	0	A	X	DFLG46
B3,G1=0	1	1	63	G1,2,3,4	0	3	0	A	X	
B3,G1=1	1	1	73	G1,2,3,4	0	5	0	A	X	DFLG46
B4,G1=0	1	1	64	G1,2,3,4	0	3	0	A	X	
B4,G1=1	1	1	74	G1,2,3,4	0	5	0	A	X	DFLG46
B5,G1=0	1	1	65	G1,2,3,4	0	3	0	A	X	
B5,G1=1	1	1	75	G1,2,3,4	0	5	0	A	X	DFLG46
BE	1	1	76		0	1	0	0	I	
BF	1	1	77		0	1	0	I	R	
CD	0	0	78		0	1	0	0	I	
CE	0	0	79		0	1	0	I	R	

(continued)

3.2.8 (Cont.)

Output Trunk

This trunk is 64 bits wide. It transmits output data to the Register File. The data formats for 32-bit and 64-bit mode are as shown below. Data will remain on this trunk for one cycle.

```
                0          15 16                    63
               -------------------------------------
64-Bit Mode   | exponent     |      coefficient     |
              |              |                      |
               -------------------------------------

                0    7 8        31 32 39 40       63
               -------------------------------------
32-Bit Mode   |expo-|coefficient|expo-|coefficient|
              |nent |           |nent |           |
               -------------------------------------
                                 \                 /
                                  --------v--------
                                   copy of 00-31
```

Output Control Trunk

The output control trunk transmits control or fault bits associated with results generated by Scalar Floating Point. These signals come up with data and are held up for one cycle. The following signals are transmitted on the output control trunk:

Signal Meaning of a "1" on Signal Line

Branch Cond. Met The operands meet the compare condition. This line is zero when a compare is not being done.

Exit Cond. Met The operands do not meet the compare condition. This line is zero when a compare is not being done.

Divide Timing Divide operands will follow

Pulse this timing pulse by 14 cycles.

Divide Busy The divide element cannot accept new operands during the time this signal is a "1".

(continued)

3.2.8 (Cont.)

Signal	Meaning of a "1" on Signal Line
Data Flags 39, 41, 42, 43, 45, 46	See specification 10354636 for these definitions.

Instruction Conflicts

Due to the various instruction cycle times, conflicts may arise at the output of Floating Point and within the unit. Floating Point operations must not be initiated on cycles which will cause conflicts. The following procedure can be used to determine these conflict cycles:

C = the cycle at which operation A is A initiated.

L = the number of cycles operation A spends in A Floating Point.

C = the cycle time at which operation B is B initiated.

L = the number of cycles operation B spends in B Floating Point.

If operation B is initiated after operation A then
$C_B \neq C_A + L_A - L_B$ to avoid a conflict.

In addition it must be remembered that no divide instruction may be initiated if the busy time has not expired from a previous divide.

3.2.9 Trace Register

Register file address zero is used as the trace register. The trace register contains the address from which the most recent branch was taken. Register zero can be referenced by executing a 7D instruction. See the instruction specification for the mode of the 7D instruction which will move register zero to Main Memory. The maintenance station can read register zero by storing the Register File and reading absolute zero from memory. After a job to monitor exchange, the job's address zero in memory contains the address of the last branch taken prior to the exchange operation. After a monitor to job exchange, monitor's address zero (absolute zero) contains the address of the last branch taken prior to the exchange operation.

3.2.10 Exchange Operation and Interrupts

The purpose of the exchange is to change the prime role of the CPU. In job mode, job tasks are performed; in monitor mode, the system decisions are made.

Some instructions in progress may be interrupted prior to their completion. The invisible flags stored in the invisible package are used to restart the interrupted instruction exactly where its output left off.

There is a package of data called the "invisible package" that contains the state of a job whenever the job is suspended from execution. If the job makes a supervisor call, or if the job suffers a fatal error (illegal instruction for example), or if an I/O Unit requires attention, the job state is placed in the invisible package and monitor mode enabled. Instructions in progress are allowed to complete.

The invisible package is always stored starting at bit address 104000_{16}. This is as indicated in the Exit Force instruction write up in the Instruction Specification.

The monitor must set up the invisible package for each job. There is NO invisible package for the monitor program itself.

If a job is to be re-entered, the monitor should not alter the job's invisible package.

Figure 3.2-2 shows the format of the invisible package.

(continued)

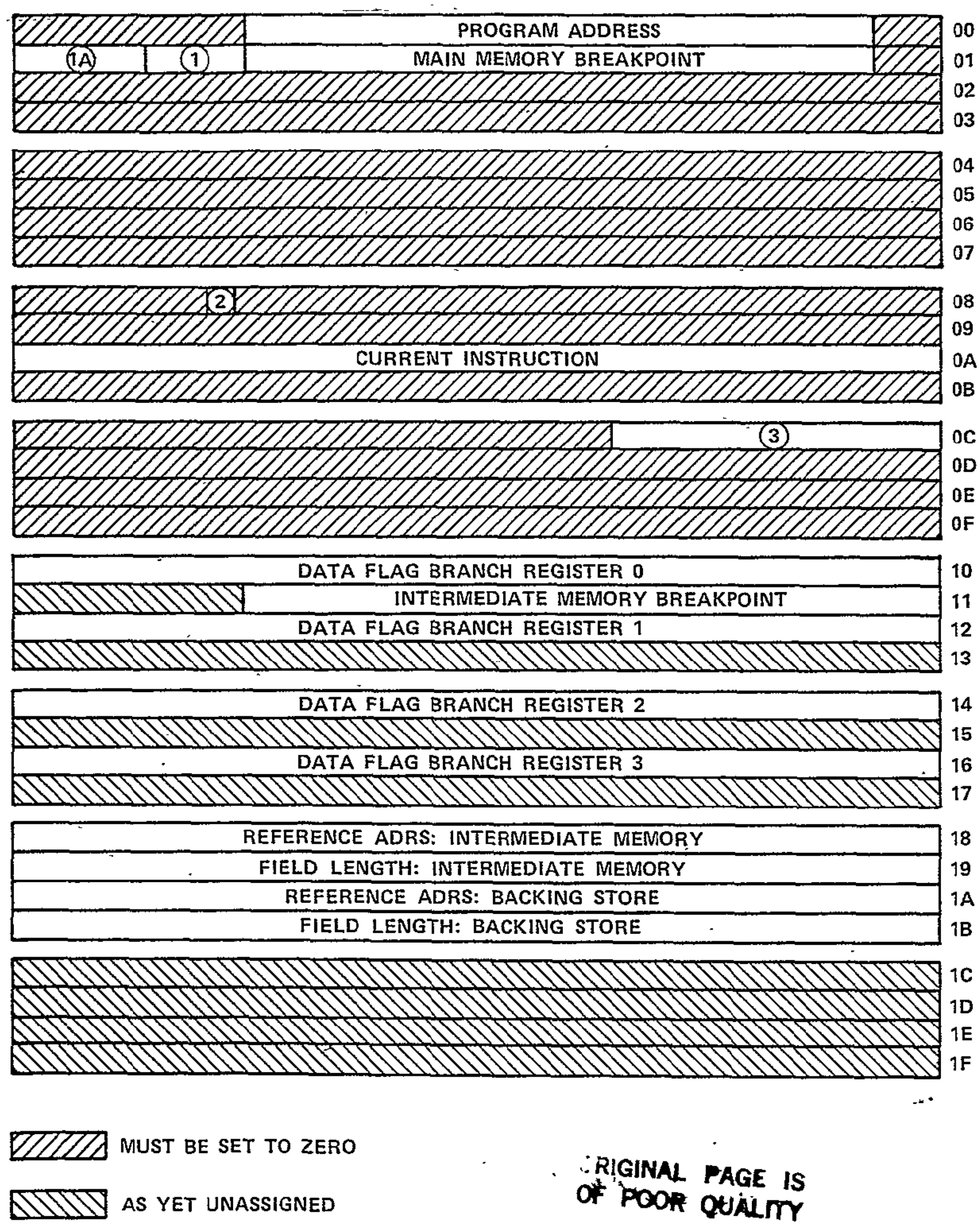


Figure 3.2-2 Invisible Package Format

3.2.10 (Cont.)

The following notes apply to Figure 3.2-2.

1 Usage bits for breakpoint registers.
2 Bit 14 Monitoring counters enable. For further information see Section 3.11 in this specification.
3 Job Interval Timer.

Exchange from the Monitor to a Job

This is always accomplished with an Exit Force instruction. The monitor program must set up the invisible package for the job prior to exchanging to that job via the Exit Force instruction. The Exit Force operation is as follows:

1. The CPU's invisible registers and flags are loaded from the invisible package located starting at the bit address 104000_{16}.

2. The Register File for monitor is stored into absolute memory locations 0 through $3FC0_{16}$ (bit addresses). The Register File for the job is loaded from the job's memory locations $100000\text{-}103FC0_{16}$ (bit addresses). Any job mode references to this area of a job's memory causes the executing instruction to be treated as an illegal instruction.

3. The CPU mode is changed from monitor mode to job mode.

4. The contents of P (program address register) are then read up and an appropriate start sequence is executed.

(continued)

3.2.10 (Cont.)

Exchange from Job to the Monitor

The Exit Force instruction and the channel interrupt are the two normal ways of getting from a job in the job mode to the monitor program in monitor mode. Attempting to execute a monitor-type instruction in job mode or an attempt to execute an undefined op-code comprise the third way into the monitor. Except for the starting point in the monitor program, the operations performed in getting to the monitor are identical for the three.

The operation is as follows:

1. The current invisible registers and flags are stored into the invisible package in memory locations 104000_{16} - $1047C0_{16}$ (bit addresses).

2. The Register File for the job is stored in memory locations $100000\text{-}103FC0_{16}$ (bit addresses) and memory locations 0 through $3FC0_{16}$ (bit addresses) are read and put into the Register File.

3. The CPU mode is changed from job mode to monitor mode. Any external interrupts which occur after this point are honored only if the CPU executes an Idle instruction. If the CPU does not execute an Idle instruction, the interrupts are saved until the CPU mode reverts to job mode, or until the monitor program clears those interrupts with a 0E (Translate External Interrupt) instruction.

4. The monitor program is executed starting at the absolute address contained in the right-most 48 bits of the monitor's register 3, 5, 6, or 7.

   Refer to Table 3.2-4 for methods of getting from job to monitor mode.

(continued)

3.2.10 (Cont.)

If an attempt is made by the monitor program to perform an undefined op-code, an automatic branch is made to the absolute address contained in the monitor's register 4. This hardware trap is to aid in the debugging of the monitor software and to trap some hardware failures. This trap is not to be utilized by the monitor software as a "normal" branch.

TABLE 3.2-4. JOB TO MONITOR METHODS

Method of Getting to the Monitor	Monitor Register, the Contents of which is Used to Set P
1. Undefined instruction, Monitor type instruction in Job Mode, or a reference to the Register File as memory (bit address $0000\text{-}3FFF_{16}$).	Register 3
2. Undefined OP Code in Monitor or reference to the Register File as memory (bit address $0000\text{-}3FFF_{16}$).	Register 4
3. Exit Force.	Register 5
4. Channel Interrupt.	Register 6

(continued)

3.2.10 (Cont.)

The bits in the external interrupt register are assigned as shown in the following table:

3.2-5. EXTERNAL INTERRUPT REGISTER BIT ASSIGNMENTS

External Interrupt Line	Assignment
0	I/O channel 0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	I/O channel 13
14	Swap Unit
15	Monitor Interval Timer

3.3 Streaming Control Unit

The Streaming Control Unit serves as the main control element for the streaming units - Vector Streaming Unit/Vector Ensemble, Main and Intermediate Map Units. See Figure 3.3-1. Streaming control has two main functions:

1) Providing instruction buffering between scalar instruction issue and the executing units. Without this buffering the Scalar Unit could not overlap scalar instructions with streaming instructions sufficiently.

2) Providing the streaming units with a means of coordinating their operations, e.g., the Vector Streaming Unit checking to see whether or not the Map Unit has completed moving data that the Vector Streaming Unit needs for a data source. The means of this coordination is a set of flags called the dependency and interlock key flags that act as semaphores between streaming instructions.

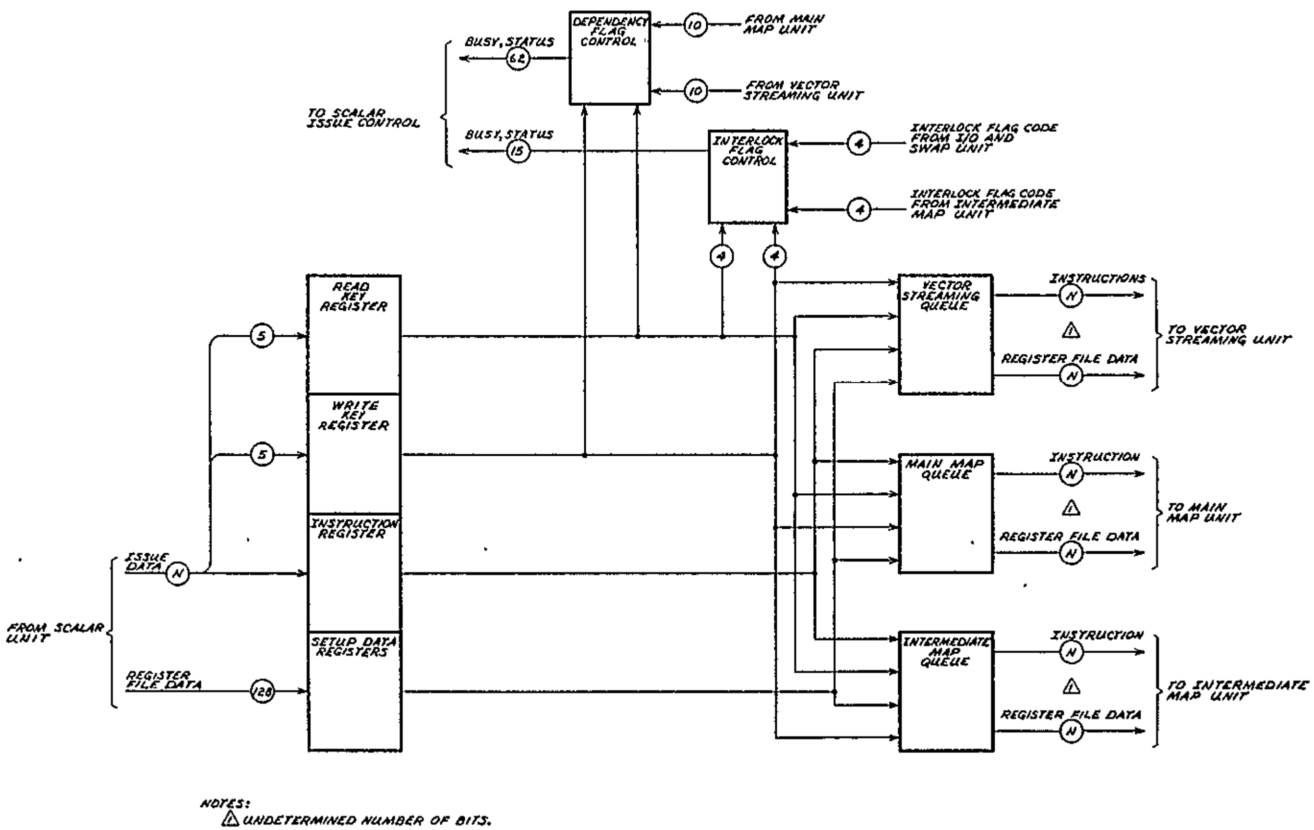


Figure 3.3-1. Streaming Control Unit

3.3.1 Instruction Entry to Streaming Control

When the Scalar Unit issues a streaming instruction, the instruction and its associated data are sent to a set of registers in the Streaming Control Unit. The information sent includes the following:

1) The instruction itself. This determines what instruction queue the instruction will enter. Various subfields within the instruction determine how the read and write key registers are checked.

2) Memory address pointers that come from the Register File. The pointers will be used by the executing instruction to control the read and write ports used by the instruction.

3) A read key. This field can be applied to check either the dependency flags or the interlock flags. How, and what is checked is determined by the instruction.

4) A write key. This field, like the read key, can check either the dependency or interlock flags under instruction control.

If a flag conflict exists, as determined by the instructions, the whole register set is locked, the instruction is prevented

(continued)

3.3.1 (Cont.)

from entering the proper instruction queue, and a signal is sent back to instruction issue control to prevent issue of further streaming instructions. Note that this has no effect on subsequent scalar instructions; they will continue to issue until another streaming instruction is to be issued. If the conflict has not cleared by that time, issue is halted until the conflict has cleared.

It should be understood that even if scalar instruction issue stops, any streaming instructions previously issued continue to execute.

3.3.2 Dependency and Interlock Flags

The key flags are used by competing resource units to coordinate their activity. The dependency flags are used by the Vector Processor and the Main Map Unit to check for data conflicts in Main Memory. The interlock flags are used by the Intermediate Map Unit, the Swap Unit, and the I/O Units to check for data conflicts in Intermediate Memory. Because the 9D instruction is used to control both the Main and Intermediate Map Units, a subfield must be decoded to see if a map instruction read or write key field refers to the dependency flags or to the interlock flags or both. The Swap and I/O Units are passed key numbers in control messages which are left in Intermediate Memory.

There are 32 dependency flags and 16 interlock flags. The ratio of these numbers reflects the expected relative activity for Main and Intermediate Memories. Flag 0 in both flag sets is held clear and this functions as a null key.

3.3.2.1 Interlock Flags

The interlock flag register consists of 15 flags which may be test and set. The flags are tested separately by the read and write keys if required by a map instruction. The flags are also used by the Swap Unit and I/O Unit through a different path. When a key is to be applied, the 4-bit key is decoded to

(continued)

3.3.2.1 (Cont.)

one of 15 flags (zero is a null). If the flag is clear, the flag is then set and approval is sent back to the requesting element. If the flag is set, the request is refused and nothing further happens. The read and write keys should not contain the same flag number unless both keys are zero (no flag check).

For a map instruction the keys are both sent to the Map Unit along with the instruction. When the Map Unit has finished wrting data for an instruction, the write key is sent back to the interlock flag register (assuming of course, that Intermediate Memory was being written) to clear the flag. In a like manner, the read key is returned.

3.3.2.2 Dependency Flags

The dependency flag register consists of 31 flags which may be test and set. However, the register is more complex in use than the interlock flags.

Each flag in the dependency flag register shall consist of three bits as follows:

Bit 1 - a read reference to this flag by the Vector Streaming Unit;

Bit 2 - a read reference to this flag by the Main Map Unit;

Bit 3 - a write reference to this flag by either the Main Map Unit or the Vector Streaming Unit.

The read key sets either bit 1 or bit 2 depending on the instruction. Just as in an interlock flag reference, a key of zero means no check or set of the flag register is made (the key is applied to a null flag).

The following rules apply to the use of the dependency flags by map and vector instructions.

a) A read key of an instruction will hold up the insruction from entering the execution queue if either the write bit of the requested flag is set, or the read bit of the rquested flag for the same functional unit is set; e.g., a vector

(continued)

3.3.2.2 (Cont.)

instruction read key checks the vector read bit for the requested flag.

b) A write key of an instruction will hold up the instruction from entering the execution queue if either the write bit of the requested flag is set, or the read bit of the requested flag for the opposite functional unit is set; e.g., a vector instruction write key checks the map read bit for the requested flag.

c) When the instruction is ready to enter the execution queue, the proper read bit of the corresponding flag will be set, and a write bit will be set according to the key in the write dependency key field.

d) The Vector Streaming Unit will signal the Streaming Control Unit to clear the vector read bit when the read ports have finished reading the data for the instruction. The same will be done for the vector write bit when the write ports have written the last data for the instruction into Main Memory.

e) The Map Unit will signal the Streaming Control Unit to clear both the map read bit and the map write bit when the last data for the instruction has been written into Main Memory since map instructions have no input (read) length. (More detailed design may reveal that the read bit may be cleared a few cycles earlier than the write bit in anticipation of a completed write.)

3.3.3 Execution Queues

There are three separate instruction queues.

1) Vector Processor Queue.

   This queue holds both the 9E and 9F instructions and associated data.

2) Main Map Queue.

   This queue holds the part of the 9D instruction to do with the Main Map Unit.

(continued)

3.3.3 (Cont.)

3) Intermediate Map Queue.

This queue holds the intermediate map part of a 9D instruction.

When a streaming instruction has cleared the flag tests, it is sent to the proper queue or queues. Each queue is a FIFO buffer of 16 words. When a functional unit is ready to take another instruction, it takes the front entry in the queue, and any instructions behind in the queue move up one position.

The 9E and 9F instructions are put into the queue in that order because the 9E instructions deal with reading operands and the 9F instructions deal with writing operands. Because of anynchronous control, the read ports are released by an insruction and are ready to set up for another instruction before the write ports are finished with an instruction.

Map instructions are of three types depending on where executed:

a) Main Map only,
b) Intermediate Map only,
c) Both Main and Intermediate Map required.

Type a) instructions are sent to the main map queue; type b) similarly, are sent to the intermediate map queue. An instruction of type c) is broken into two segments and each segment is put into its respective queue with an identifying flag attached. When an instruction reaches the front of a map queue with a flag set, the instruction is not sent to be executed immediately, but instead, is held up until the other map queue has an instruction with a corresponding flag in the front of its queue. Both queue entries are then sent to their respective map units.

3.4 Vector Processor

The Vector Processor consists of two functional units - the Vector Ensemble and the Vector Streaming Unit. The Vector Ensemble performs floating-point arithmetic on vector and array data that resides in Main Memory. The Vector Streaming Unit does the addressing of memory and the management of the resulting data streams for the Vector Ensemble.

The units of the Vector Processor operate together but each has its own control section and each is given control separately. The input section (memory read) of the Vector Streaming Unit is controlled by the 9E instruction. The Vector Ensemble and the output section (memory write) of the Vector Streaming Unit are controlled by the 9F instruction.

The Vector Ensemble consists of five identical Vector Units (pipelines). Four of the pipelines operate on data under program control leaving the fifth pipeline as a spare. The Maintenance Control Unit (MCU) designates which unit is to be the spare. If all five pipelines are operational the MCU can rotate the spare pipeline by selecting a different spare between jobs. The spare unit can be driven in parallel with another functioning unit, thus allowing the outputs of the two units to be compared and checked, or the spare can be supplied data by, and results returned to, the MCU. This last option allows a pipeline to be fixed off-line, while the FMP is still running jobs, unnoticed by users and without reduction in throughput. Each vector pipeline has several internal data cmparators that enable a pipeline to check itself (See 3.4.1.7.3). If an error is discovered by an on-line pipeline, the job in process at the time is abandoned and the MCU is notified of the error. The MCU then reconfigures the Vector Ensemble to make the pipeline with the error the spare unit, and passes control back to the FMP Operating System.

The vector pipelines operate in lockstep; that is, each unit is in the same internal state, performing the same operations, as any other unit though each is operating on separate data. Each pipeline is designed so that it accepts two sets of data on each

(continued)

3.4 (Cont.)

machine clock cycle. Thus, the four Vector Units operating together yield a result rate of eight result operands per clock cycle (64-bit mode).

The Vector Processor has four access ports into Main Memory and two result ports. Each pipeline can perform up to three floating-point operations on data passing through; thus the four pipelines operating together and in conjunction with the Vector Streaming Unit can give a true result rate of 24 floating-point results to memory each clock cycle (64-bit mode).

The Vector Processor runs under its own local control. That is, Instruction Issue in the Scalar Unit passes sufficient information to the Vector Processor via the Streaming Control Unit so that it can proceed independently. No active control is required from Instruction Issue. When the Vector Processor is given a process to perform, it checks for resources required and, if available, sets up and performs the required operations. If the resources are not available, the setup information is held in a one-word queue until the resources become available. This queue is separate from the Vector Streaming Queue that resides in the Streaming Control Unit. When the Streaming Control Unit finds the queue full, it suspends issuing to the Vector Processor until the queue is emptied.

The Vector Processor queue is, in fact, two queues - one each for the Vector Ensemble and Vector Streaming Unit. The queues are further broken down according to the separate resources of each unit. Thus, if an individual resource is available, it immediately tries to perform the desired function.

Holding its own setup information locally, a resource has two additional requirements in order to perform a function: valid data at its input and a place that will accept the processed output. This then is the control system for the Vector Processor - when valid data is presented at the input to a function resource, if the resource has been set up to perform an operation it sends an "accept" to the sending resource, and

(continued)

3.4 (Cont.)

some number of cycles later produces valid output data. If the receiving resource is able to take the data it does so. If, however, proper setup of the receiving resource has not as yet been fully accomplished, acceptance of the data is not forced. If a given resource does not receive an "accept" from the receiving resource, it stops sending "accepts" to the resource supplying its input. Valid data is indicated by a "valid" signal on a single line (called the valid line) that accompanies the data. The "accept" signal is also a single line (called the accept line).

Thus, a complete operation consists of setting up a complete "valid" chain - from things that can source "valids" (Main Memory ports) to things that can sink "valids" (also Main Memory ports). If a complete "valid" chain is established for a given operation, that operation will proceed to completion.

As an example of the above consider a vector operation being performed using two source streams (say A and B) and one result stream. If the next operation to be performed (the operation specified by the next operation setup queue) requires all four input streams and both result streams, the C and D streams of the second instruction will set up immediately and will fetch data to an internal buffer set within the Vector Streaming Unit. Because the Vector Ensemble is not looking for data from the C and D streams for the current operation, no C or D stream data will be sent to the pipelines. When the A and B streams are finished requesting data for the current instruction, they start immediately requesting data for the new A and B streams. Notice that these requests take place before the last data from memory has passed into the pipelines to be processed. If the new requests to memory can be acknowledged with no delay, this instruction may have no startup time due to memory access.

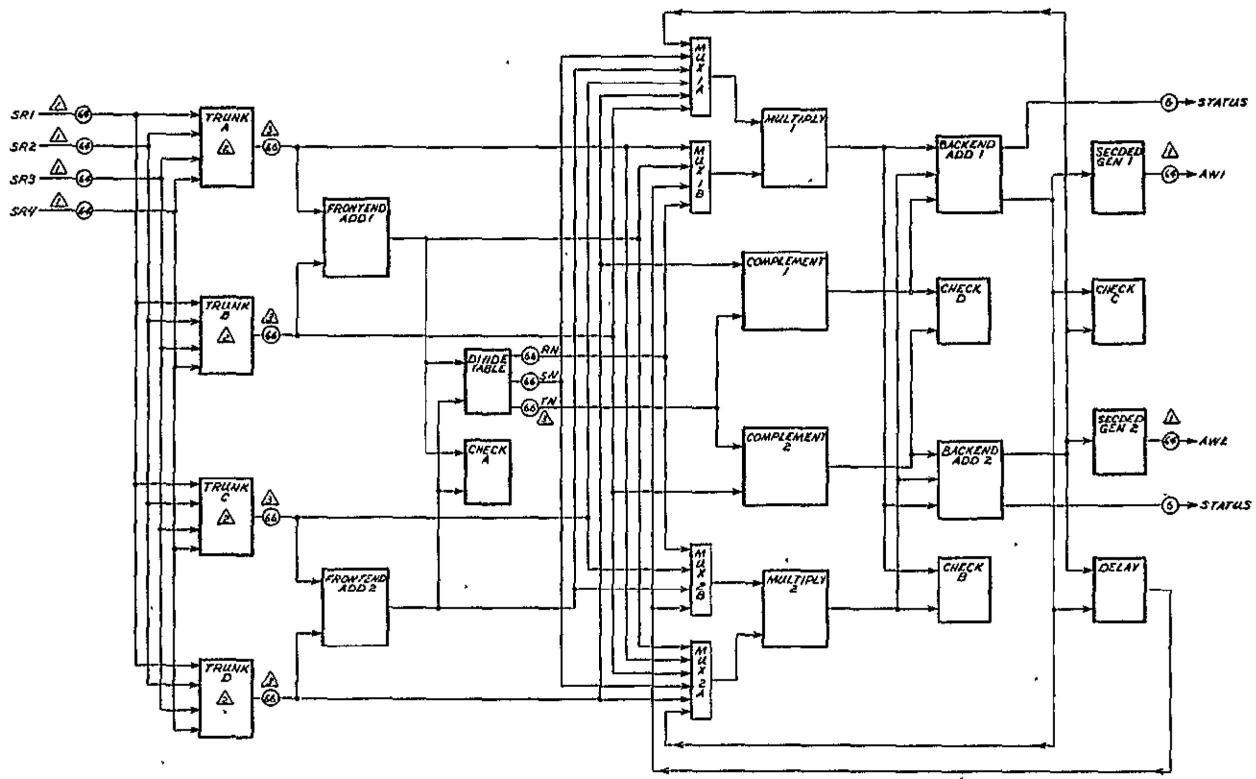
(continued)

3.4 (Cont.)

When the last of the data for the old instruction has passed into the Vector Units, pipeline control looks to see if the new instruction to be performed conflicts with the old instruction. If a conflict exists, the new data is prevented from entering the pipelines until the data is processed sufficiently to remove the conflict. When the data for the new instruction starts moving through the pipelines, the control attempts to set up the write ports in the Vector Streaming Unit. One of the output ports will set up immediately because it was unused by the old instruction. The other port setup has to wait until the last of the data for the old instruction has been passed to memory before setting up for the new instruction. This may cause a delay of one or two cycles before the setup is completed and result data from the new instruction is on its way to memory.

3.4.1 Vector Ensemble (VE)

Figure 3.4-1 provides a simplified block diagram of a single Vector Unit in the ensemble. Each unit is completely independent of another, with no interconnections between them for data or control. All incoming and outgoing control passes between each unit and the Vector Streaming Unit. Each Vector Unit contains two full multiplier and adder elements and two half-adder elements, each of which is capable of operating on pairs of 64-bit input operands or quartets of 32-bit operands every half clock cycle. Each arithmetic element (add, multiply) is a segmented pipeline, seven segments per element. Each segment requires one half clock cycle of pipeline time. Thus two operands proceeding through all three segments for a combination add and multiply ((A+B)*C) would require about ten minor cycles to pass from the select network to the result buses, Arithmetic Write 1 (AW1) and Arithmetic Write 2 (AW2). A simple, normalized ADD operation utilizes the front-end add elements, bypasses the multiply elements and completes the addition and post-normalization in the back-end add elements. The total segments for a simple, normalized ADD is seven or for a simple MULTIPLY operation it is also seven segments of pipeline time. This pipeline length contributes to vector startup time as described in section 3.12.2.



NOTES:
1. THESE BUSES CARRY SECDED (NOT SHOWN IN BIT WIDTHS).
2. EXCLUDES SECDED CORRECT NETWORK.
3. DUPLICATE SIGN BITS.
4. ALL INPUTS/OUTPUTS ARE FROM/TO VECTOR STREAMING UNIT.

Figure 3.4-1 One Vector Unit

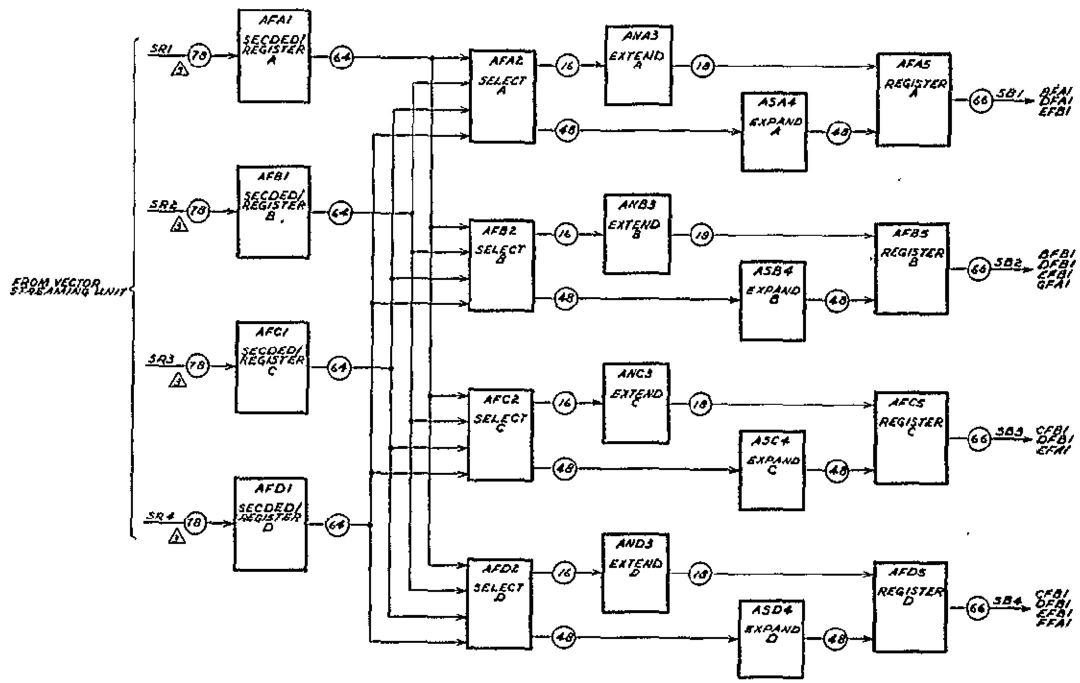
3.4.1.1 Read Bus Select Elements

There are four input data buses for each Vector Unit, SR1 through SR4. As shown in figure 3.4-2, each input bus is capable of supplying operands to any or all four of the functional streams (Trunk A, Trunk B, Trunk C, Trunk D) which feed the various arithmetic elements. As can be seen from figure 3.4-1 then, any combination of input buses can be fed to any of the arithmetic elements, permitting such combinations to occur as (A*A)+(B*B) by supplying A operands via SR1 and selecting it through MUX1A and MUX1B to both sides of the multiply element. Likewise the B operands could be supplied via SR2 and selected through MUX2A and MUX2B to both sides of the second multiply element. The results of the two multiplies would then be combined in back-end adder 1, to form the sum of the two products.

The read bus select elements Trunk A, B, C, and D are individually controlled by the A, B, C, and D fields of the 9F (Vector Arithmetic) instruction which is interpreted by Instruction Issue and transmitted to the Vector Ensemble by the Streaming Control Unit.

The extend blocks in the trunk networks perform two functions. First, the block duplicates the sign bit(s) of the input exponent(s). This is done to facilitate exponent underflow/ overflow detection and also to preserve a proper exponent in case some intermediate calculation causes an underflow or overflow, but a subsequent calculation gives a result in a legal range. Second, the block serves to detect endcase operands. Endcases are overflow/indefinite and underflow/ machine zero. When an endcase operand is detected, a signal is sent along with the operand to direct further processing of the operand.

The expand blocks provide the ability to perform mixed mode calculation within the Vector Ensemble. A calculation is considered mixed if one operand is in 32-bit mode and the other operand is in 64-bit mode. When performing mixed mode calculations the mantissa of the 32-bit operand is extended from 24 to 48 bits by appending 24 bits of zero to the lower part of the mantissa.



NOTES:
1. THE EXTEND BLOCKS DUPLICATE THE SIGN BITS, DETERMINE ENDCASES.
2. THE EXPAND BLOCKS EXPAND 32-BIT OPERANDS TO 64-BIT OPERANDS IF REQUIRED.
3. INCLUDES 64 DATA BITS AND 14 SECDED BITS.

Figure 3.4-2. Trunk Networks of Vector Ensemble

3.4.1.2 Write Bus Select Elements

On any given half-clock cycle the back-end add elements together can produce two 64-bit or four 32-bit results, each of which are placed on their respective arithmetic write buses (AW1 and AW2). The results appearing on these two buses are defined by the suboperation codes for the 9F (Vector Arithmetic) instruction. If either the V field (for AW1) or the W field (for AW2) are non-zero, the respective results are sent to the Vector Streaming Unit to be stored in memory. A field value of zero inhibits storing any data for the corresponding result stream. (See section 3.2.1.160 of the instruction spec.)

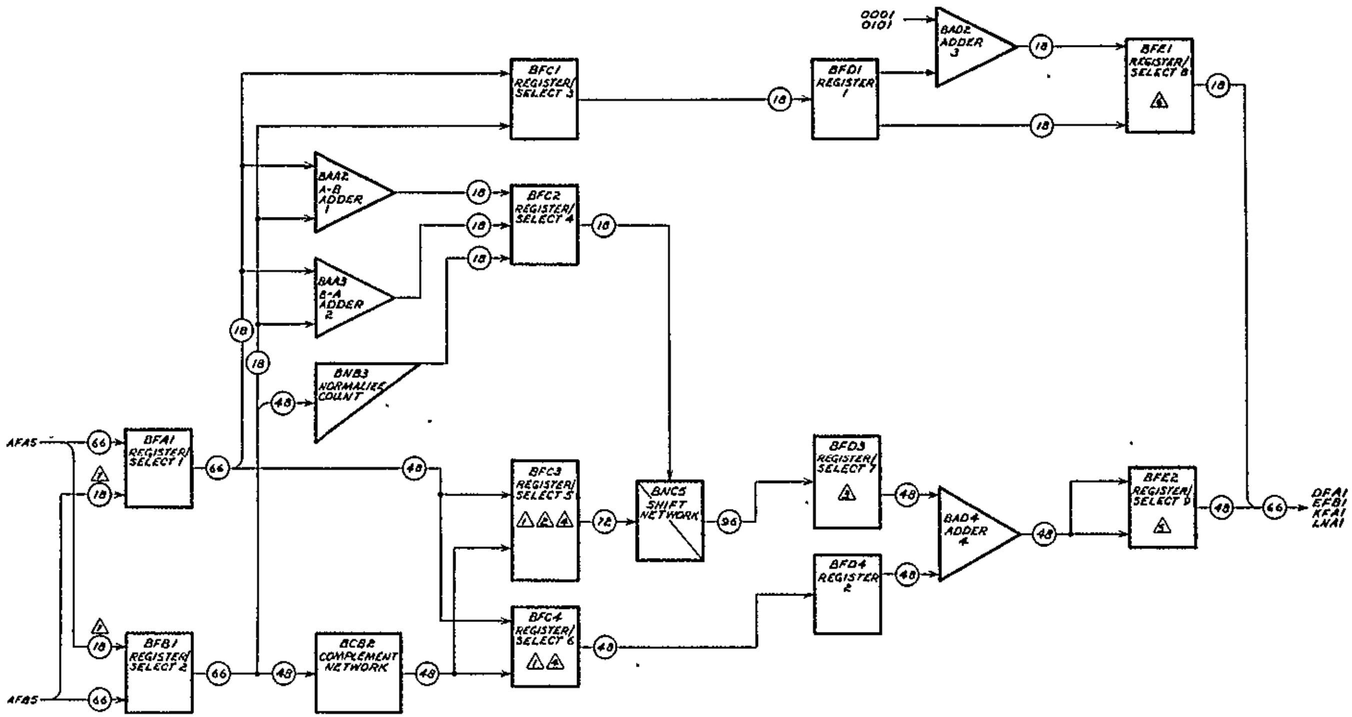
3.4.1.3 Front-End Add Elements

Two identical arithmetic elements form the front-end functional processors of each Vector Unit. Figure 3.4-3 shows a block diagram for one front-end adder. These elements are composed of a prenormalize network which aligns operands of unlike exponents, plus a full two's complement adder producing one 64-bit or two 32-bit results every half-cycle. There is no post-normalization shift network present in these elements. The output results from such an element is the equivalent of the FMP ADD or SUBTRACT UPPER or LOWER, with no normalize shifts being done on the result data.

The primary function of these adders in primitive operations (diadic arithmetic such as A*B) is to perform the pre-normalization of input operands (particularly for the divisor in divide operations) and to provide for complementing of one or more operands for functions such as (-A*B).

Each front-end add element has its own independent microcode control so that diagnostics can be loaded via the microcode trunk to perform failure isolation to the lowest replaceable component level (LSI chip).

In addition to the pre-normalization of the divisor in divide operations, these elements perform the necessary complementation of negative source operands prior to performing the table look-up that initiates the reciprocal approximations.



NOTES:
1. OPERAND SELECTED DEPENDS ON EXPONENT COMPARE RESULT.
2. 24 ZERO BITS INSERTED BETWEEN COEFFICIENTS IF IN 32-BIT MODE.
3. CATENATES OPERANDS IF IN 32-BIT MODE.
4. RIGHT SHIFT ONE IF COEFFICIENT=800...0 AFTER COMPLEMENT (AND SEND INCREMENT SIGNAL TO EXPONENT).
5. RIGHT SHIFT ONE IF COEFFICIENT OVERFLOW OCCURS.
6. SELECT INCREMENTED EXPONENT IF COEFFICIENT OVERFLOW OCCURS.
7. EXPONENT IS COPIED TO OTHER TRUNK IF IDLE.

Figure 3.4-3. One Front-end Adder of Vector Unit

3.4.1.4 Multiply Elements

Each Vector Unit contains two identical multiply elements each with its own independent control logic. The multiply element inputs two 64-bit or four 32-bit operands and produces one 64-bit or two 32-bit results every half-clock cycle. This multiply operation is performed in seven segments, each of which requires a half-cycle. In the first segment, four-bit groups of the multiplier are used to encode 8-bit groups of the multiplicand into a series of partial sums and carries. For the remaining segment times, these partial sums and carries are merged through a series of partial adders yielding a 96-bit wide product of partial sums and carries which are finally added together in the back-end adder. This addition operation produces a 96-bit wide coefficient result which can either be normalized, truncated, rounded, or left in upper or lower form (for double-precision arithemtic).

Inputs to the multiplier are controlled by the subfunction operations specified in the 9F (Vector Arithmetic) Instruction, and can come from the read bus select networks, the front-end adders, the divide table element (for divide operations), or from one of the arithmetic result buses emerging from the back-end adders depending on which operations, such as PRODUCT, are desired. If one or both of the multiply elements is not specified in the suboperation, then identical inputs are selected for both elements and checking is enabled.

3.4.1.5 Back-end Adder Elements

Each Vector Unit contains two identical back-end adder units, each with its own independent control logic. The back-end adder consists of a rank of deskew logic for synchronizing the various partial sums and carries from the multiply elements , and a full <u>three-input</u> adder capable of combining the multiply output results with the output of the complement network or the other multiplier element. This function provides facilities for functions such as (A*B)+C or (A*B)+(C*D).

Each back-end adder performs a 96-bit (in 64-bit operand mode) or two 48-bit (in 32-bit operand mode) coefficient addition every half-cycle. The first two segments perform the first

(continued)

3.4.1.5 (Cont.)

addition of a pair of operands. The second two segments contain the second addition of the resulting input pair of operands plus the final group carry/generate tree, and the final segments contain the rounding/truncation logic and post-normalization network. Post-normalization is controlled by the type of operation specified in the 9F instruction.

Inputs to back-end adders are controlled by the subfunction code in the 9F (Vector Arithmetic) instruction. The outputs are placed on the arithemtic write buses AW1 and AW2 by the back-end adders.

3.4.1.6 Divide Table Element

The following description is done for a 64-bit divide operation. A 32-bit divide proceeds in an identical manner except that different bits are taken from the operand coefficient and only the first pass through the Vector Unit is required to obtain a proper result.

The divide operation utilizes most of the arithmetic elements in the Vector Unit. To achieve a divide rate of one result per half-cycle (for 24-bit coefficient accuracy), the reciprocal divide approximation is utilized. In this mode, the divisor is pre-normalized and its absolute value yielded by a front-end adder. This resulting divisor is then sampled by taking 11 bits of the coefficient from the left-most (or most significant) end, not including the sign (which will always be zero since the absolute value of the divisor is used), and not including the most significant bit (which will always be one since a normalized divisor is used), yielding bits 18-28 of the 64-bit operand. These eleven bits are used to address a read-only memory (ROM), or look-up table, called the divide table element. A 39-bit word (plus one parity bit) is read from the ROM at that address. The word is partitioned into two fields, S (14 bits) and T (25 bits). The field is used as input to the other front-end adder (for complementation if the divisor was originally negative), and the S field is used as input to a multiplier to form the product of S times the remaining bits of

(continued)

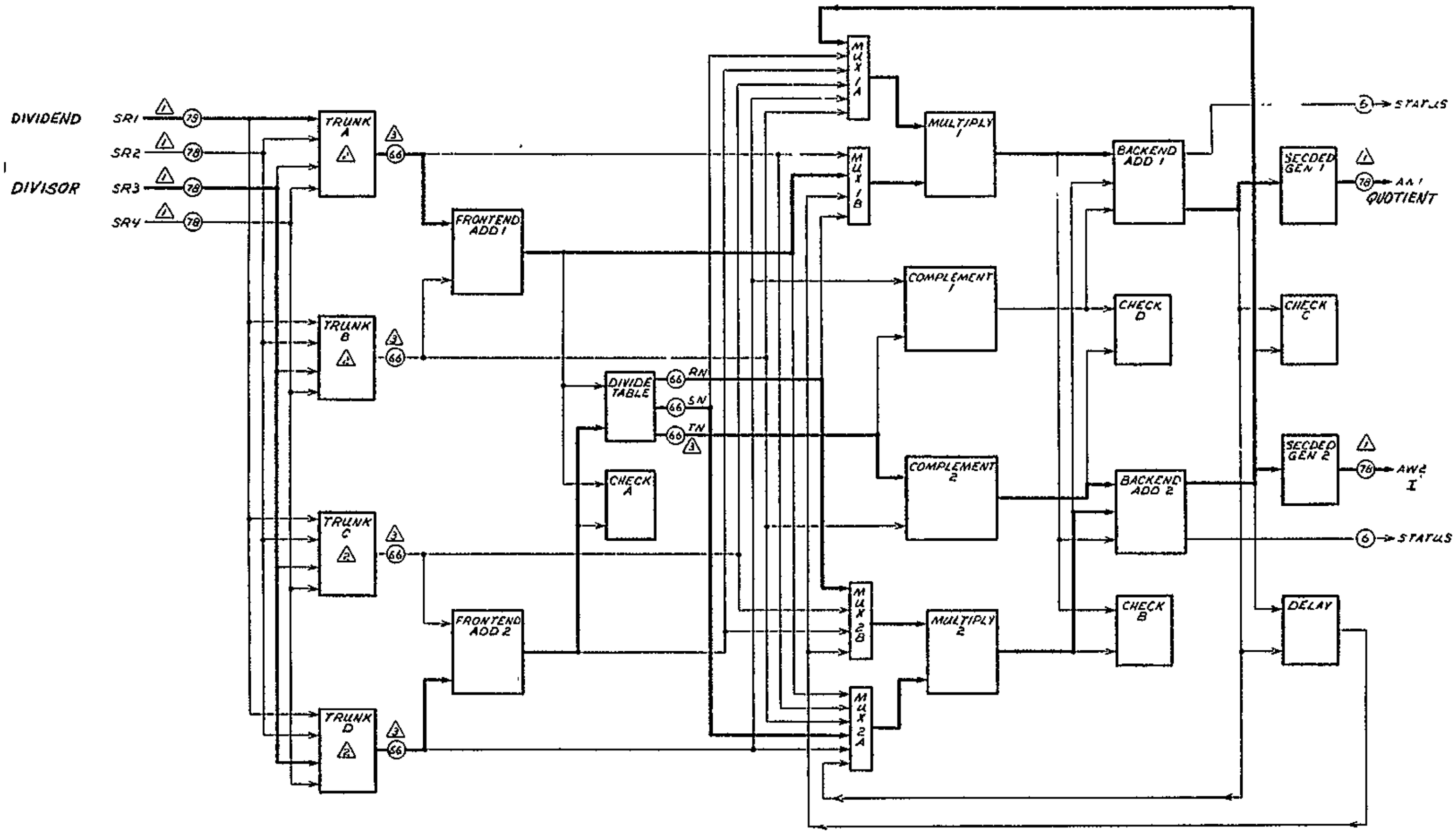
3.4.1.6 (Cont.)

the coefficient (the 35 bits not used in the table look-up). The multiplied result is subtracted from T in the back-end adder and that result is then fed into the other multiplier along with the dividend to form a 64-bit result of which at least 23 bits of the coefficient are accurate. The pair of results out of the back-end adders can be stored in memory, thence retrieved for a second pass divide operation to perform the necessary corrections to produce a full 64-bit result, accurate to 47 places.

Figure 3.4-4 shows the interconnection for the first pass divide operation which yields a result correct to 23 or more bits.

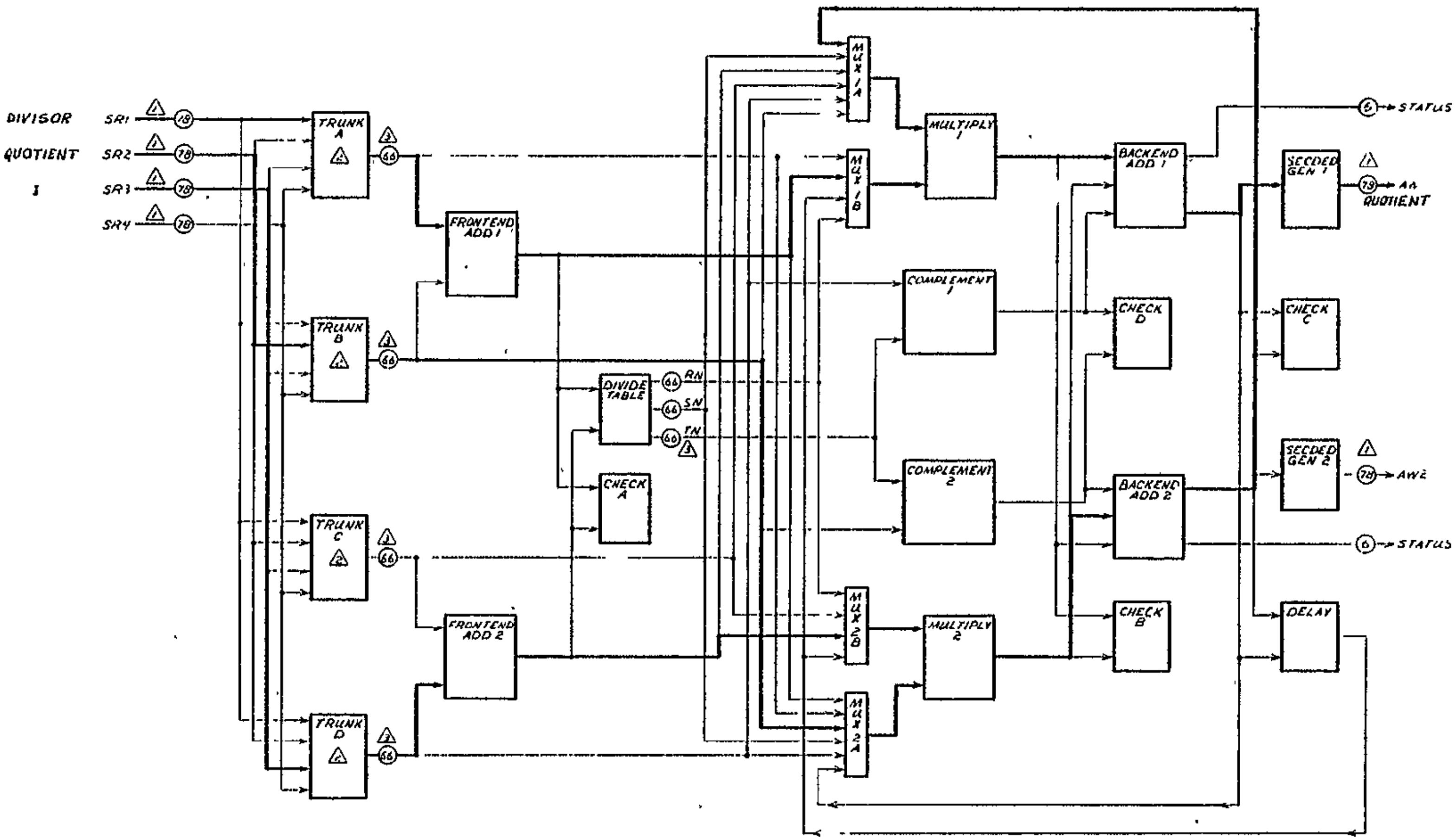
Figure 3.4-5 shows the interconnection scheme for the second pass divide operation which is used to produce 64-bit floating-point quotient results. The input operands required for this second pass are: the first pass quotient (which is by itself adequate for 32-bit arithmetic), the original divisor, and the intermediate product.

The divide table element is referenced once each half-cycle during the first pass divide operation. This means that when in 32-bit mode, the divide rate is the same as for 64-bit mode during the first pass, one result per half-cycle. Usually however, the need for 48-bit accuracy in the coefficient portion of the 64-bit result will require the second pass which then creates an effective 64-bit divide rate of one result every two half-cycles per Vector Unit.



NOTES:
1. INCLUDES SECDED CHECK BITS.
2. INCLUDES SECDED CORRECT NETWORK.
3. DUPLICATE SIGN BITS.
4. ALL INPUTS/OUTPUTS ARE FROM/TO VECTOR STREAMING UNIT.

Figure 3.4-4 First Pass for 32-Bit Divide



NOTES
1. INCLUDES SECDED CHECK BITS.
2. INCLUDES SECDED CORRECT NETWORK.
3. DUPLICATE SIGN BITS.
4. ALL INPUTS/OUTPUTS ARE FROM/TO VECTOR STREAMING UNIT.

Figure 3.4-5 Second Pass for 64-Bit Divide

3.4.1.7 Error Checking

3.4.1.7.1 SECDED

Each of the trunks entering and leaving a Vector Unit and connecting to the Vector Streaming Unit contain a SECDED (single error correction, double error detection) networks. The read buses (SR1-SR4) contain SECDED detection and correction circuits, while the write buses (AW1-AW2) contains SECDED code generation networks.

SECDED is carried on a 32-bit basis, seven bits for each 32 bits of data. Thus all input and output trunks possess 78 actual bits of transmitted data.

See section 3.11.5 for additional information on SECDED.

3.4.1.7.2 Parity

The divide table element consists of a loadable random access memory (RAM) that behaves as a read only memory (ROM) during normal Vector Unit operation. Each 39 bits of divide table data have a single parity bit associated with them. Upon each table read, the parity is checked. If an error occurs, the Vector Unit is immediately halted and the Maintenance Control Unit (MCU) is alerted by an error flag. In addition, the Scalar Unit is sent a stop signal.

Upon command of the MCU the Vector Unit can transmit the failing memory location in the divide table, the operand location in the input vectors for the failing case, and the P counters of all the control microcodes for the Vector Unit, to assist in maintenance actions.

Each of the microcode memories in the pipeline control networks contains a parity bit for each word addressed. In the event that a parity error occurs, the microcode sequence is frozen and the P counter transmitted to the MCU on command. A flag indicating which microcode is failing is sent to the MCU. The Scalar Unit is also sent a stop signal.

3.4.1.7.3 Result Checking

Each Vector Unit is supplied with four coincidence checking networks, capable of comparing the results produced by the identical pairs of arithmetic elements. CHECK A compares the outputs of the front-end adders, CHECK B compares the outputs of the multipliers, CHECK C compares the results of the back-end adders, and CHECK D compares the outputs of the complement networks. Checking is enabled under the following circumstances.

1. When the same input trunks are selected into the pair of operand ports A&C and B&D, and the identical functions are selected for the pair of elements (front-end adders, multipliers, back-end adders, or complement networks).

2. When a given element pair is idled during an operation. For example, the suboperation code 04 would invoke the operation A*D and B*C thus idling the complement networks. In this case an operand is taken from the divide table and is enabled into both complement networks automatically by the Vector Unit. The output, athough meaningless to the programmer, would be checked by the checking network.

3. When one of a pair of elements is idled by a particular suboperation code. For example the suboperation code 08 would cause the operation A*C+D, thus one multiplier would be idle. In this case the Vector Unit would automatically enable the same pair of inputs to both multiply elements. The checker would then be enabled.

It can be seen that the programmer can explicitly control checking in some cases by setting the appropriate fields in a 9F add instruction to select identical operands to identical elements. See appendix E for details of when checking is enabled.

In the event that an enabled checker discovers a mismatch in the output data, the Vector Unit is halted, a stop signal is sent to the Scalar Unit, and the MCU is alerted.

3.4.1.8 32/64-Bit Arithmetic

Each Vector Unit is capable of processing two 32-bit or one 64-bit results each half-cycle in each of its arithmetic element segments, except for the divide table which produces one 32-bit result per half-cycle.

Each arithmetic element except for the divide table can also process a combination of one 64-bit and one 32-bit operand each half-cycle as input to an operation. For example, a front-end add element could be accepting a 64-bit input operand on its A trunk and a 32-bit operand on its B trunk. In this mixed mode a 64-bit result would be produced.

Each of the input trunks from the Vector Streaming Unit provides a flag indicating what mode that particular trunk is operating in, either 64 or 32-bit. The Vector Unit then automatically configures its arithmetic elements to accept that form of data on that trunk.

The 9F instruction provides the size mode for the output trunks.

(continued)

3.4.1.8 (Cont.)

Floating-point numbers in the CDC FMP are two lengths, 32 bits and 64 bits as shown in the figure below. The 32-bit format has an 8-bit exponent and a 24-bit coefficient. The 64-bit format has a 16-bit exponent and a 48-bit coefficient. The left-most bit of each exponent and coefficient is the sign bit. A detailed description of floating arithmetic is presented in the instruction specification.

```
                                                  32-BIT FORMAT

0    7 8               31 32  39 40              63
----------------------------------------------------
|      |                 |      |                  |
| (8)  |      (24)       | (8)  |      (24)        |
|      |                 |      |                  |
----------------------------------------------------
\__  _/ \_______  _______/ \__ _/ \______  ________/
   v           v             v           v
UPPER      UPPER          LOWER      LOWER
EXPONENT   COEFFICIENT    EXPONENT   COEFFICIENT
```

```
                                                  64-BIT FORMAT

0          15 16                                  63
----------------------------------------------------
|            |                                     |
|    (16)    |                (48)                 |
|            |                                     |
----------------------------------------------------
\_____  _____/ \_______________  __________________/
      v                        v
   EXPONENT               COEFFICIENT
```

3.4.1.9 Asynchronous Control

As all other units, the Vector Unit controls the movement of operands through its various elements by valid/accept signals. Therefore, as soon as data valid signals appear at the ports selected by a particular 9E/9F operations, the Vector Unit will begin to move the data through its networks provided no resource conflicts exist. The results will be placed on the output buses with a valid signal, and no more data will be placed there until an accept is received from the trunk destination. That is the purpose of the fields in the 9F which designate output ports on which to expect accepts during an arithmetic operation.

Likewise, the Vector Unit returns an accept for every operand it takes from an input port, thus allowing the port supplying operands to move a fresh operand into place on the trunk. In the case of mixed mode operations where the rate of supply can exceed the rate that the Vector Unit can process, the accept flag consists of two bits indicating whether the lower or upper 32-bit operand has been accepted on the particular 64-bit trunk. See Appendix B for a more detailed discussion of asynchronous data movement control.

3.4.1.10 Control Signals

(To be defined later)

3.4.1.11 Microcode Terms

(To be defined later)

3.4.1.12 Interface Timing

(To be defined later)

3.4.2 Vector Streaming Unit

The purpose of the Vector Streaming Unit is to control and manage the data streams for the Vector Ensemble. Figure 3.4-6 is an overall block diagram of the Vector Streaming Unit (VSU).

The responsibilities of the VSU include the following:

- Providing addressing to Main Memory for the 6 data streams (4 read streams and 2 write streams). This includes incrementing the addresses properly as the vector operation proceeds, as well as determination that a particular data stream has supplied all the required data for a particular vector operation.

- Performing physical alignment of the resulting streams of data. This alignment is required in two places--on input data, the proper elements of the input streams must be matched no matter what the source address of the data; on output data, the results from the Vector Ensemble must be returned to the proper addresses in Main Memory.

- Performing temporal alignment of the input data streams. Because of memory access delays, as well as some other factors, corresponding data words may not be available from memory simultaneously. Also for a large set of possible vector operations the individual input data streams are not required at the same time, but instead must be offset from one another by precise amounts. For example, in the operation R=(A*B)+C, A and B are used simultaneously but C must be available to meet the A*B product at the proper time.

- Pipeline switching. In the event that an active pipeline fails, the spare pipeline must be brought into play by sending streaming data to the spare and taking its results in place of those from the failing pipeline.

(continued)

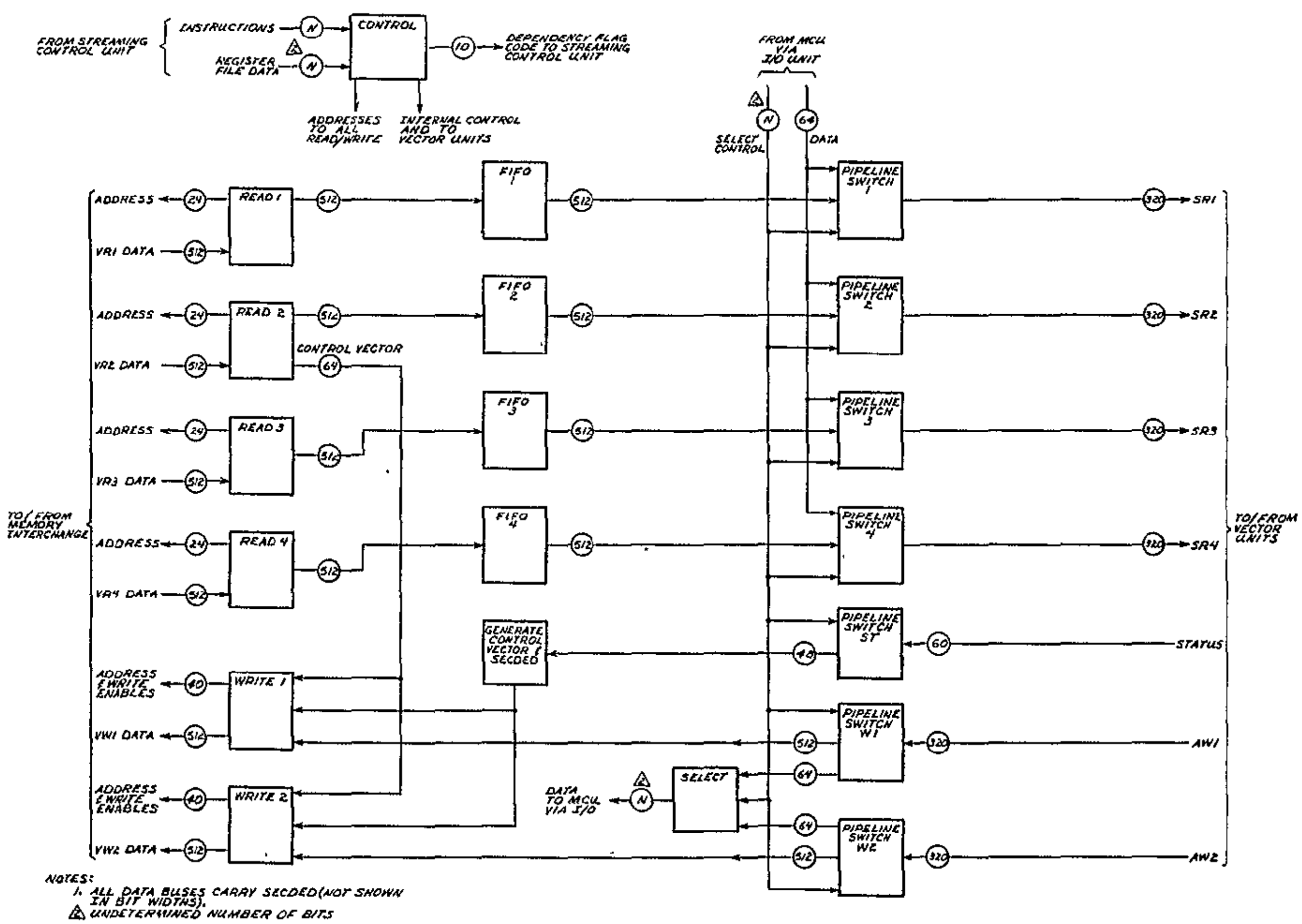


Figure 3.4-6 Vector Streaming Unit

3.4.2 (Cont.)

- Bit vector generation. The programmer may want to generate a bit vector where each bit in the bit vector represents the result of some comparison with a data word in a data stream. For example, bit R(I)=A(I).GT.(0.0001). The bit vector can be used then by both the Vector Streaming Unit (see below) and the Map Unit.

- Bit vector interpretation. The programmer may want to use a bit vector whose purpose is to inhibit certain vector results from being stored. As shown in the figure below, say that one row of an array holds initial conditions for some problem but the rest of the array is to be updated by some vector process. If a vector of bits of the form 100...00100...00100... is used to control whether or not a particular result is to be stored in memory (here a "1" means do not store), the updating of the result array can be done with one vector instruction rather than several shorter vectors which have greater total setup overhead.

```
                 ----------------------
Memory  |  |//////////////////////|  -- Initial
Store   |  |----------------------|     conditions
order   |  |                      |     not to be
        v  |                      |     altered
           |                      |
           |                      |
           |                      |
           ----------------------
```

The VSU is controlled in the same manner as the other streaming units. That is, the control scheme involves distributing control functions among the various elements to be controlled. Each element is given enough information to allow it to run asynchronously from the other elements. In this way the maximum number of events may happen in parallel without the need for a very complex central control unit. See appendix B for a more detailed discussion of asynchronous data movement control.

3.4.2.1 Read Ports of VSU

Figure 3.4-7 shows a block diagram for one of four nearly identical read ports within the VSU. Each read port controls a separate data stream from Main Memory to the Vector Ensemble.

The read ports receive the following setup information to control the data streams:

1 The address of a vector stream to be fetched.

2 The number of data words to fetch in the stream (the input length).

3 The number of words of data that the output stream will require (the output length). This output length will, of course, be the same for all read ports.

4 The word size (mode) of the resulting data stream.

5 Exception information.

3.4.2.1.1 Port Memory Addressing

When the read port receives the memory address of the requested data, the port enters the address into a counter/register and immediately starts making memory requests. Each clock cycle thereafter the address is incremented and the memory request control makes another request to memory for more data. As the memory address is being incremented, both the input and output lengths are decremented. The amount subtracted on each cycle from each length counter depends on the operand mode (8 for 64-bit mode and 16 for 32-bit mode).

If the output length counter becomes equal to or less than zero, then further requests to memory for this data stream are stopped and the request control loads an address and starts requests for the next vector. If the input length count becomes equal to or less than zero before the output length reaches zero, what happens next depends on exception condition data. There are basically two ways to continue: either repeat the vector data stream from the beginning or fill the remaining vector elements with a constant.

(continued)

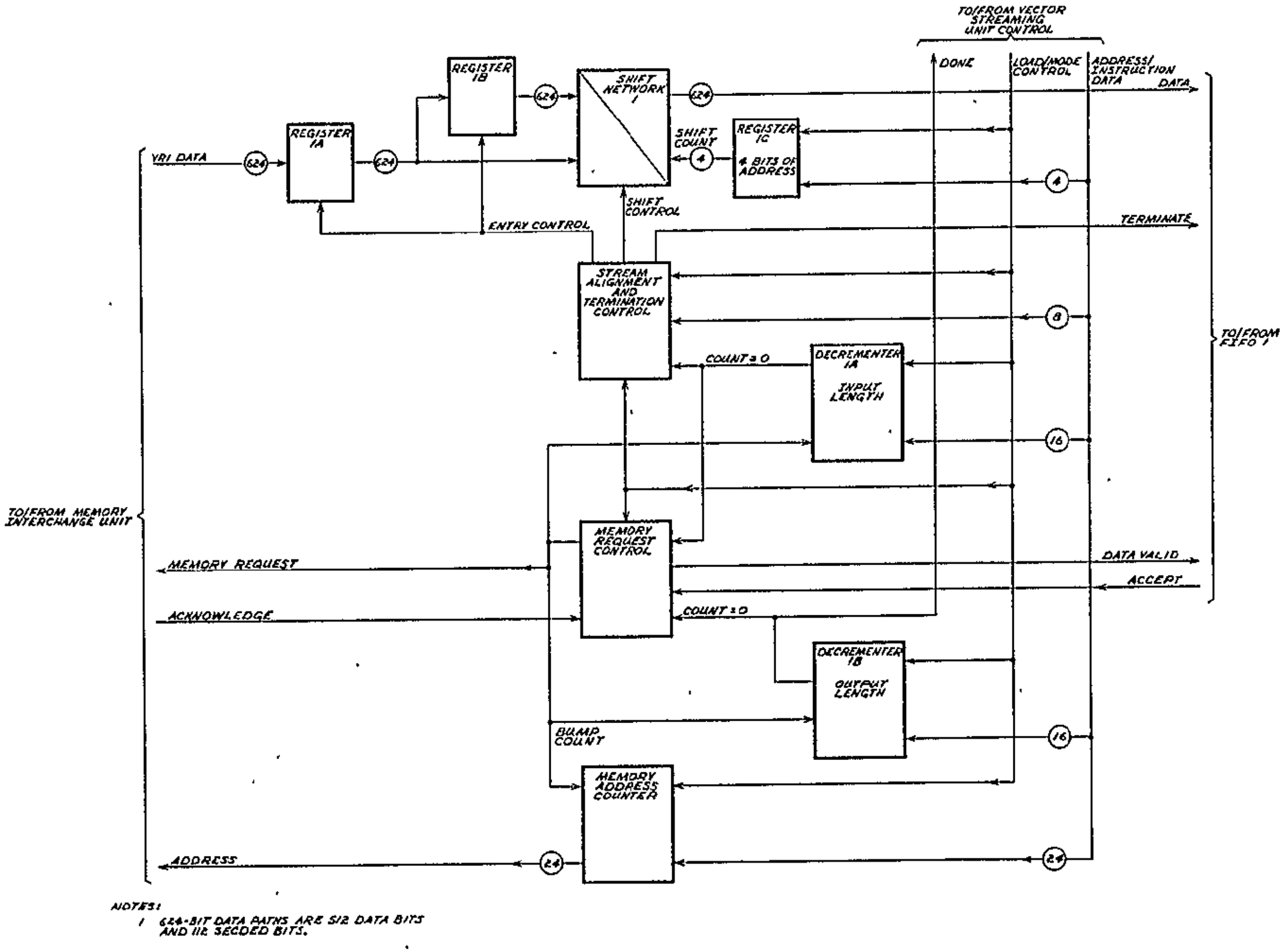


Figure 3.4-7. One Read Port of VSU

3.4.2.1.1 (Cont.)

In the first case the memory address counter is reset to its original address and memory requests are continued. This reloading of the address counter is repeated as many times as required until the output length is satisfied. This feature is useful for a case like the following (shown in FORTRAN):

```
      DIMENSION VEC(50),ARRAY(50,50)
      DO 1 J = 1,50
      DO 1 I = 1,50
      ARRAY(I,J) = ARRAY(I,J)+VEC(I)
  1   CONTINUE
```

Using the automatic repeat feature, the above loop can be executed in one long vector stream instead of several (50) shorter ones.

In the second case, providing a constant to provide fill of the remaining elements, the read port provides three options as to the constant: either a floating-point zero, a floating-point one, or floating-point indefinite. This last option is the normal mode because there is seldom a need to add a shorter vector to a longer one.

In the above discussion no mention was made of streaming data actually being received by the port because, for a short, normal vector, the addressing logic can have completed requests to memory for one stream and started on another vector without having "seen" data from the first vector. Information that is required by the port data logic is entered into a first in, first out (FIFO) queue to await data from memory. This is an asynchronous process because the amount of time from a read request to memory until the data arrives is not fixed.

Other than satisfying a length count, two conditions, both of a similar nature, can interrupt the read port from making memory requests:

1. The addressing control keeps a count of the number of requests for memory data that are outstanding (have not been acknowledged) if the count reaches a predefined limit (about 8 requests), memory access requests are held up.

2. The addressing control can be notified by the data FIFO buffer control (see 3.4.2.2) that the buffer is in danger of overflowing. In this case also, memory access requests are halted until the condition is cleared.

3.4.2.1.2 Port Data Management

The data arriving at the port from Main Memory is in "RAW" form. That is, the 512 bits at the port are ordered exactly as they were fetched from memory. 64-bit word 0 of the arriving data was taken from a module 0 (modulo 8); likewise word 1 from memory came from a module 1, etc. Because vectors have no restriction on starting address, it is very unlikely that two different data streams will matchup if both are used "RAW".

The read port performs the required physical alignment of the data. During the time of the first fetch of data from Main Memory, the port computes a shift count that will be required to align the data stream so that the first element of the vector will be sent to pipeline 0. As streaming data arrives, it is shifted the required amount and sent on its way. Notice that this means some data fetched by the first request may be thrown away because it was "in front of" the first element of the required vector.

Because vector lengths are uncontrolled, it is unlikely that the last vector element of a particular vector will go into the last pipeline. Thus, extra data will have been obtained on the last fetch of a vector. This data is sent along with the other data with a flag attached to the last valid word going to a particular pipeline. This flag has two purposes. First, on ordinary instructions, the flag notifies individual pipelines to start setup for the next instruction. Second, on recursive instructions (SUM for example), the flag serves to notify the pipelines to start the termination sequence. The flag joins the data in entering the FIFO buffer.

3.4.2.2 FIFO Function

The FIFO buffer in each input data stream serves to align the input stream operands with respect to time. There are two phases to the problem - facilitating alignment with respect to Main Memory and facilitating alignment with respect to vector pipelines.

A data stream arriving at the vector pipeline at the same time as a required companion stream may seem an obvious requirement. There are, however, some difficulties that mitigate against this happening.

1. The asynchronous nature of the control structure for the Vector Processor means that there is no specific relationship between the vector ports when they make requests to memory for vector streams.

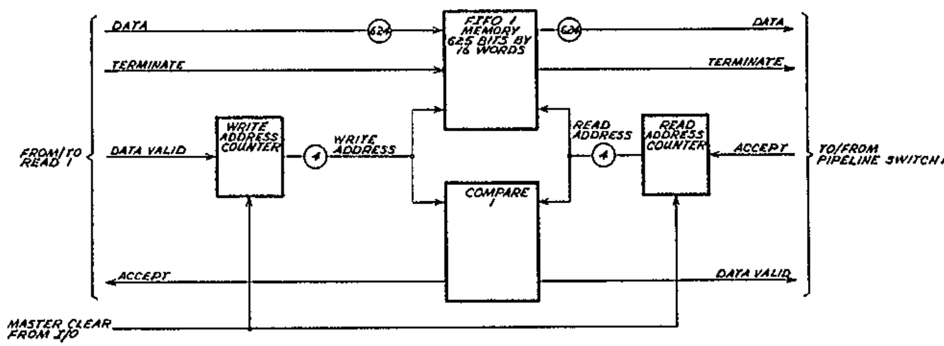
2. Other memory access requests may conflict with a memory request from a vector port, thus causing memory busy delays in receiving data. It may even happen that the vector ports conflict between themselves by making simultaneous (or nearly simultaneous) requests to the same set of memory modules. This problem is made worse because one of its side effects is to reduce the effective memory bandwidth.

What is needed, then, is a buffer that allows data to reside close to the Vector Units, holding the data until all the required streams are available to be sent to the pipelines while allowing delayed streams to "keep their place in line" by continuing to make requests. The data accumulates in the buffers until the vector pipelines start using the data. At that point, the read ports are filling the buffer at the same rate that it is being emptied, keeping the number of words in the buffer constant until the end of the vector is reached. While the data at the end of the vector is being taken by the pipelines, the data ports can be starting the next vector. This may result in data from more than one vector coexisting in the buffer. The termination flag attached by the read port to the last element in the vector serves to separate the vectors.

(continued)

3.4.2.2 (Cont.)

Figure 3.4-8 presents a block diagram of one FIFO buffer. Functionally, it is a random access memory of about 16 words by 512 bits (plus SECDED bits). Two separate addresses are kept for the buffer, a read address, controlled by accepts from the vector pipelines, and a write address, controlld by valids from the read port. When the FMP is initially started, the address pointers are set equal to each other (e.g., each is set to zero). A comparator continuously compares the two addresses and produces a signal, called "empty", if the addresses are equal. If the write address is one less than the read address, the comparator provides a signal called "full". As data is written into the buffer, the address pointer is incremented; if the last highest address of the buffer is incremented, the address is reset to zero and counting continued (end-around). When the buffer has data (the comparator says "not empty"), a valid signal is sent to the pipelines. As long as the comparator says "not full" the FIFO will respond with an accept to every valid received from the read port. If the comparator says "full" accepts to the read port stop until the FIFO again becomes "not full".



NOTES:

1. 624-BIT DATA PATHS ARE 512 DATA BITS PLUS 112 SECDED BITS.
2. THE COMPARE NETWORK WILL RETURN ACCEPTS AS LONG AS THE FIFO IS NOT FULL AND WILL SEND DATA VALIDS AS LONG AS THE FIFO IS NOT EMPTY.
3. THE FIFO IS A FUNCTIONING RAM MEMORY THAT IS MADE TO FUNCTION AS A FIFO BY CONTROLLING THE READ AND WRITE ADDRESS.

Figure 3.4-8 One FIFO Buffer of VSU

3.4.2.3 Pipeline Selection

Each of the four FIFO buffers in the VSU is connected to the Vector Ensemble by a pipeline switch which performs two primary functions:

1. Multiplex data together, taking the eight 64-bit operands or sixteen 32-bit operands and sending the data to the four functioning pipelines at a double clock rate (each operand is valid for a half-clock cycle).

2. Provide the ability to replace a failing pipeline with a spare pipeline.

Figure 3.4-9 gives a block diagram of the interconnection of one FIFO buffer in the Vector Streaming Unit and the Vector Units. As can be seen from the figure, there are actually connections to five physically distinct Vector Units comprising the Vector Ensemble; these connections are labeled DATA 0 through DATA 4. Only one of the input trunks is shown here, labeled WORD 0 through WORD 7, corresponding to the source trunk VR1 of the VSU. Also shown is a special data trunk labeled MCU data, which can be selected into any or all of the five physical Vector Units. The selection of maintenance data in and maintenance data out is under the control of the Maintenance Control Unit (MCU).

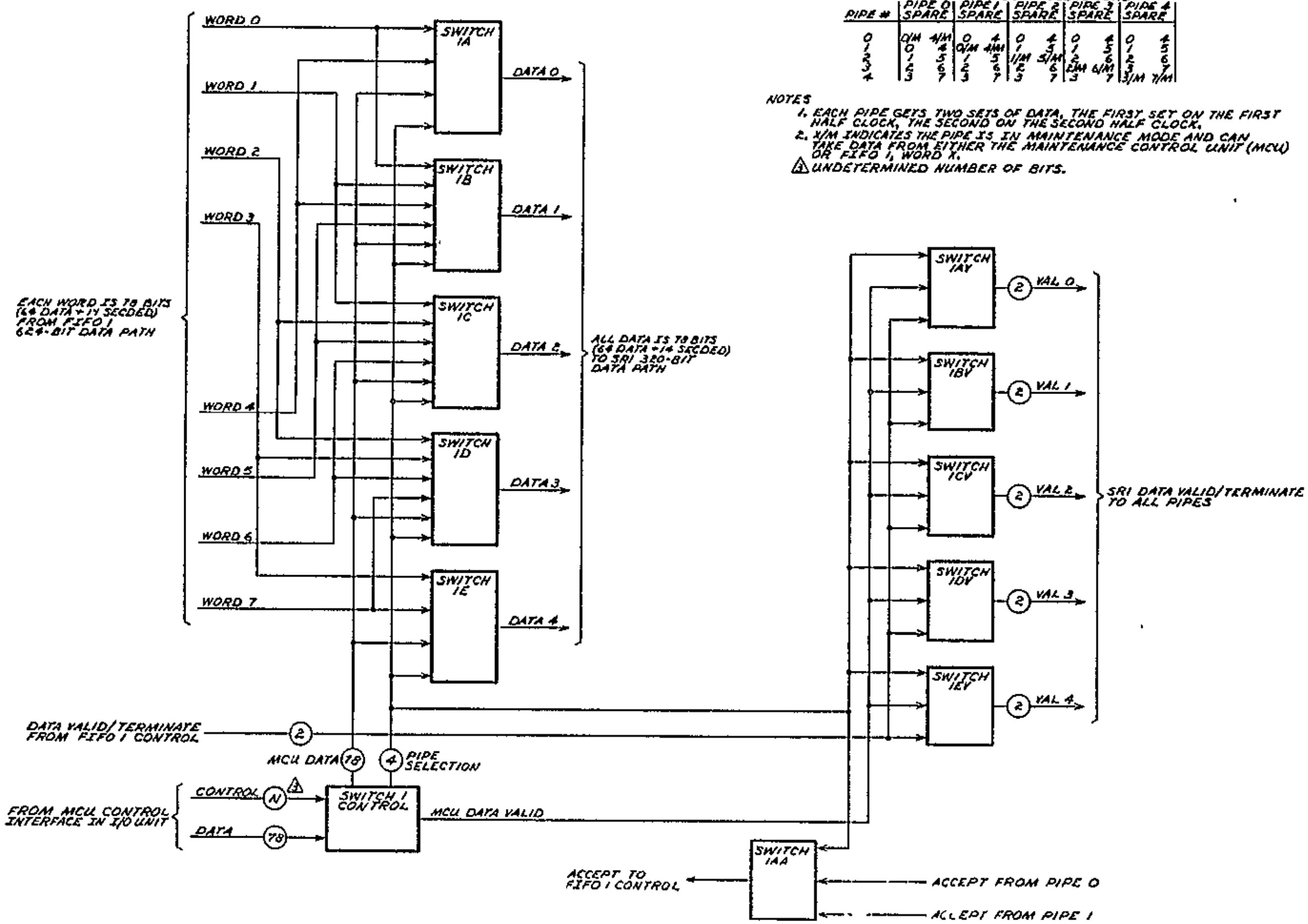
The figure represents only one of four identical pipeline switches, one for each of the four buses from Main Memory to the Vector Ensemble. In a similar manner (also not shown in the figure), two pipeline switches serve to select four of the five Vector Units (isolate one) to connect to the two write ports to Main Memory.

The write pipeline switches perform generally the inverse operations of the switches in the read paths:

1. Demultiplex data from the pipelines back to the write data port.

2. Provide the write portion of the logic which enables use of a spare pipeline.

(continued)



DATA INPUT TO EACH PIPE FOR DIFFERENT SPARE PIPELINE SELECTIONS

PIPE #	PIPE 0 SPARE	PIPE 1 SPARE	PIPE 2 SPARE	PIPE 3 SPARE	PIPE 4 SPARE
0	0/M 4/M	0 4	0 4	0 4	0 4
1	0 4	0/M 4/M	1 5	1 5	1 5
2	1 5	1 5	1/M 5/M	2 6	2 6
3	2 6	2 6	2 6	2/M 6/M	3 7
4	3 7	3 7	3 7	3 7	3/M 7/M

NOTES
1. EACH PIPE GETS TWO SETS OF DATA, THE FIRST SET ON THE FIRST HALF CLOCK, THE SECOND ON THE SECOND HALF CLOCK.
2. X/M INDICATES THE PIPE IS IN MAINTENANCE MODE AND CAN TAKE DATA FROM EITHER THE MAINTENANCE CONTROL UNIT (MCU) OR FIFO 1, WORD X.
3. UNDETERMINED NUMBER OF BITS.

Figure 3.4-9 Pipeline Selection/Isolation

3.4.2.3 (Cont.)

The write switches take results from the four active pipelines plus results from the spare pipeline and "spread" the two sets of results per clock cycle into eight 64-bit data sets plus maintenance information for the MCU.

In order to generate the eight results from four pipelines, the first four results from the first half-cycle are held in a register within the switch until the second half-cycle when four more results arrive; the data is then sent to the write port.

3.4.2.3.1 Normal Operation of Selection Networks

Upon deadstart of the FMP, the Maintenance Control Unit (MCU) sets up the input data selection networks and output data selection networks for four pipelines. Normally the pipelines would be configured with Vector 0 through Vector 3 on-line to the input and output data trunks. In addition, the data trunk of the adjacent Vector Unit would be enabled to the extra Vector Unit (in this case Vector 4). The output of Vector 4 would not be selected into WRITE 1 (VW1), but could be sampled by the Maintenance Control Unit. Thus during execution of vector arithmetic instructions, Vector 4 (in this example) would be performing identical operations on data identical to that submitted to Vector 3. Thus the internal arithmetic elements and checking circuitry of the excess unit are continuously exercised.

In the event that the excess unit discovers an error in its own operation (checker failure, parity error or SECDED double error), the Vector Unit will be halted but no stop flag will be sent to any other units. The Maintenance Control Unit (MCU) will be alerted, however. Under control of the MCU, special data trunks can be connected to the input and output of the excess unit and fault isolation diagnostics executed with selected data being forced into the trunks of the failing unit. This technique permits the on-line maintenance of a failing Vector Unit.

3.4.2.3.2 Error Recovery and Maintenance

In the event that an error is detected in one of the on-line Vector Units, the entire FMP is halted and the job in progress is aborted. Before another job is started the MCU will switch the data bus selects so that the excess unit is introduced into the system, and the failing unit removed. For example, if Vector 2 were to fail and thus be switched off-line, the input selects would be changed so that WORD 2 and WORD 6 would now go to Vector 3, WORD 3 and WORD 7 would now go to the previously off-line Vector 4. At the same time the output selects would be changed in similar manner, as well as maintenance communications enabled with Vector 2 through the data buses.

This scheme permits the use of any Vector Unit as the excess unit, depending on the MCU controls set up, thus all pipelines can be continuously exercised in an on-line manner thoughout the operating day. In such instances, the Maintenance Control Unit could rotate the assignments between jobs.

3.4.2.4 Write Ports of VSU

Two write ports are provided in the VSU each taking one result stream from the corresponding write pipeline switch.

3.4.2.4.1 Addressing

The setup information received from the Streaming Control Unit includes three principle elements:

1. The base address for the result.

2. The number of operands to be stored.

3. The result mode (32-bit or 64-bit operands).

When results are available from the vector pipelines (the port can be set up before the vector data has entered the vector pipelines under some circumstances) the port sends an address along with the data to be stored. As each request is sent to Main Memory the storage address is incremented and the output length count is decremented by an amount determined by the mode (16 for 32-bit mode, 8 for 64-bit mode). When the length count gets to, or below zero, the output port is ready to start setup for the next instruction.

3.4.2.4.2 Data Management

Data to be written into memory is accepted by the memory port 512 bits at a time, with word 0 (64 bits) always going to module 0 (modulo 8). However, since vector starting addresses have no restrictions, the data arriving from the pipelines will quite likely require alignment. This requires that the VSU write port shift the data so as to put the first result in the proper position in the output stream. Notice that the result of this is that not all 512 bits from the data bus are written into memory on the first store for a vector operation. This implies that some of the data from the pipelines is held in the write port from cycle to cycle because the pipelines produce 512 bits of result per cycle but the first store does not take all 512 bits. On the last store, in a like manner, the 512-bit bus may not be full.

The mechanism for telling the Memory Interchange if a particular result is to be stored or not is called the write enable. If no write enable accompanies a result, then the result is not stored into memory. This is done on a 32-bit basis, i.e., each 32-bit half-word has a write enable associated with it. The requirement for this can be seen from the discussion above about the first and last storage request on a vector operation. Once this ability is available, it is not very difficult to give the programmer the ability to control the write enables to inhibit storing unwanted results in memory. Read port 2 is given the ability to fetch a vector of bits from memory and to send them to the write ports where they are used to inhibit or enable the storing of particular results into Main Memory.

The vector pipelines also have the ability to generate control vectors. The write ports can take this bit data and accumulate it 8 or 16 bits at a time, depending on the mode, and store it in memory. Bits are accumulated into 512-bit swords before being stored. All generated bit vectors are aligned onto a 32-bit boundary and the last 32-bits of the bit vector are zero filled as required to round it up to the nearest 32-bit boundary.

3.5 Map Units

The Map Units (Main Map and Intermediate Map) are used to reorder (Map) data from one form to another, for example to transpose an array. The Map Units can each operate with its respective memory or, operating together, can map data from Intermediate Memory to Main Memory or vice-versa.

The map units can each perform the following:

Function	Purpose
Gather	Take noncontiguous data and "put it together" so that it can be used as a contiguous vector.
Scatter	Take contiguous data and "scatter" it back in memory in a noncontiguous form (inverse of Gather).
Compress	Pass over selected elements of an input vector, forcing unwanted elements from the result vector.
Mask	Combine two vectors - the result vector element is the top element of one of the input vectors; the vector element not chosen is thrown away. The vector element chosen is under the control of a bit vector.
Merge	Combine two vectors - take the result vector element from the top element of one of the input vectors. The vector element not chosen is kept (corresponds to shuffling cards).

See the description of the 9D instruction in instruction specification 10354636 for more detailed explanations.

The Map Units operating together can perform the Gather and Compress operations with source data in Intermediate Memory and the result data being stored in Main Memory. The units operating together can perform the Scatter operation with the source data in Main Memory and the result data being stored in Intermediate Memory.

3.5.1 Main Map Unit

Functionally, the Map Unit is very simple; as shown in figure 3.5-1, it consists of 3 read ports to fetch operands, functional units to perform the required operations and a write port to store the results into Main Memory. Connections to the Intermediate Map Unit can take the place of some of the read ports or the write ports when the two units work together.

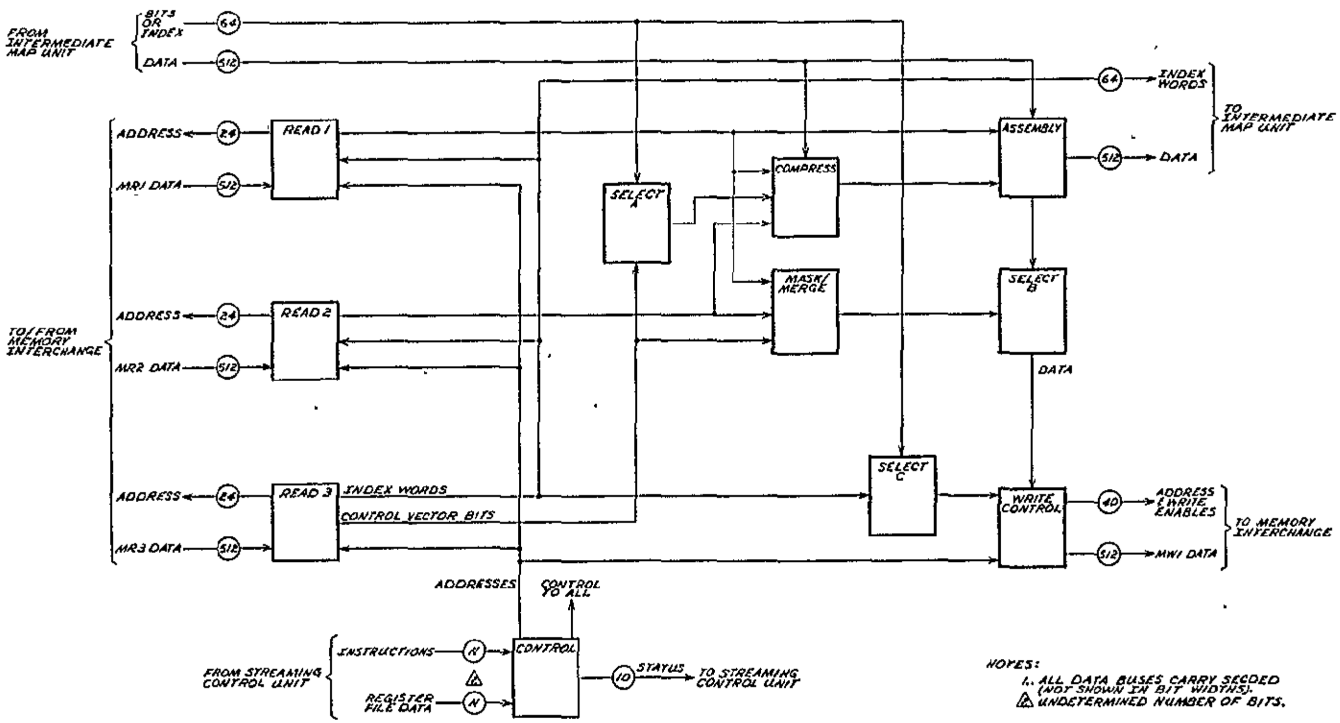


Figure 3.5-1 Main Map Unit

3.5.1.1 READ 1 and READ 2

The READ 1 and READ 2 data ports are identical internally and so will be described together. The read ports each have internal control so as to enable them to function asynchronously as much as possible.

3.5.1.1.1 Address Management

Each port is set up with information that includes the base address of the stream to be fetched and information about how and when to change the address. Each port has its own control so that when it receives sufficient setup information it immediately starts making memory requests.

3.5.1.1.2 Data Management

As data arrives from Main Memory it must be shifted so that the first word of the requested data appears left-most in the data stream. This alignment is required by all the using elements.

3.5.1.2 READ 3 Port

The READ 3 port is much like the READ 1 and READ 2 ports except that it supplies index lists and bit vectors instead of normal data. Index lists are used to generate memory addresses for some options of the Gather and Scatter instructions; bit vectors are used to control the operations of Compress, Mask, and Merge.

The additional functionality for READ 3 includes data management of the index list (whether the list is from memory or specified in a "stride" suboperation) and bit vectors (from memory). Extra control is required in data management to shift the incoming data properly because index list data is used two words at a time and control vector data is used 8 bits at a time. Thus, both modes require data to be held in the port and parcelled out for several cycles before requesting another sword of data.

3.5.1.3 COMPRESS Element

In a Compress operation, data is passed from source to result with some of the data "compressed out". The data kept or deleted is specified by a control vector (a vector of bits) where each bit in the control vector is associated with a corresponding word in the input stream. Compress control is told in its setup whether the operand size is 32 or 64 bits. See figure 3.5-2 for a detailed block diagram of the COMPRESS Element.

The complexity of Compress is such that the 8 input operands are broken in two groups of 4 and the groups then undergo compress separately. After the data is compressed it must be accumulated into 512-bit groups and the groups sent to be written into memory. The data to be accumulated is in 3 parts:

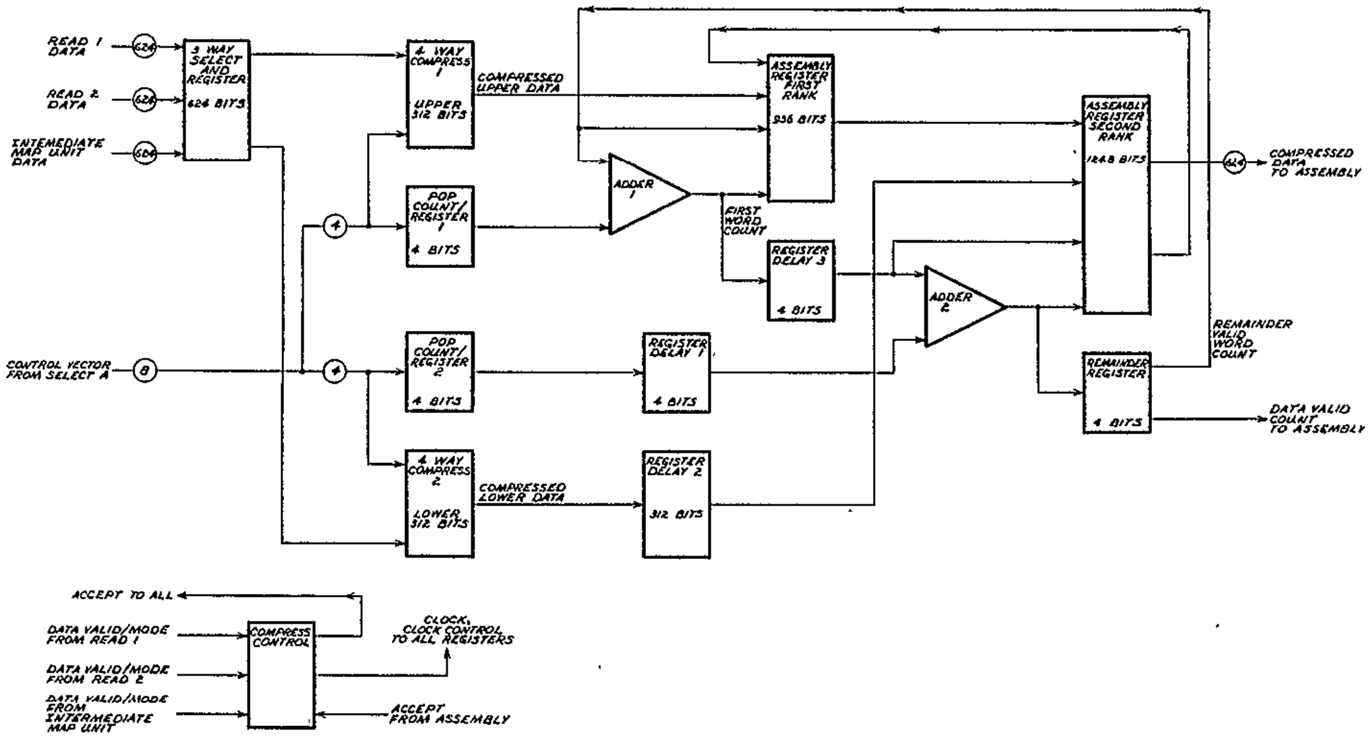
1. - A partial result from the last clock cycle (group of 480 bits or less) to be combined with Compress result of current clock cycle,

2. - The Compress results from the upper 4 operands, and

3. - The Compress results from the lower 4 operands.

The partial result must be left-adjusted and the upper result placed to the right of the partial result. The lower result must then be placed to the right of the previusly combined results. This is accomplished by two shift registers which are controlled by a count of the number of valid operands in each separate part. The count is the number of enable bits in the bit vector.

After the data groups are combined, the count of total number of valid operands is determined. If a total of more than 512 bits is valid, the left-most 512 bits of the result is sent to be written into memory and any remainder data is recycled back to meet the Compress results of the next cycle.



NOTES:

1. 624-BIT DATA PATHS ARE 512 DATA BITS AND 112 SECDED BITS.

Figure 3.5-2. Compress Element of Main Map Unit

3.5.1.4 MASK/MERGE Element

The Mask operation is very simple: For the ith result select either A(i) or B(i); this is a set of simple OR gates. The result of the operation is sent to be stored into memory.

The Merge operation is another logically complex one. Each bit in the control vector chooses a result element from either the A or B stream inputs. However, unlike the Mask operation, the unused operand is not discarded but kept for possible selection on the next cycle. The difficulty comes not when one result is selected in a given clock cycle but when 8 results are selected in a clock cycle. Each stream is processed separately to combine them into a final result.

Each stream is put into a set of controllable shift registers. What is done is to insert a fill operand (null) for each place that the bit vector indicates that the other stream will have an operand inserted. One operand stream will use the "1s" of the bit vector to insert nulls; the other stream will insert nulls using "0s". As an example, assume two streams A(1) through A(4) and likewise, B(1) through B(4) along with four bits of control vector C = 1011. Assume that the A operands insert on zeros and the B operands insert on ones. Then this will result in:

A = A(1), Null, A(2), A(3)
B = Null, B(1), Null, Null

The modified streams are then sent on to the mask logic which selects from the input operands according to the control vector.

3.5.1.5 ASSEMBLY Element

The ASSEMBLY element is a word controlled shift register that is principally used to hold partial results from the Gather operation until 512 bits of result are accumulated to be sent to the write port. If the Gather operation specifies long records, greater than 512 bits, the ASSEMBLY element simply passes the data through.

3.5.1.6 WRITE Port

The Main Map Unit WRITE port controls the storing of data to Main Memory. Data accepted by the WRITE port is all 512 bits wide plus appended SECDED.

Address management by the WRITE port is done in either of two modes. The first mode is the normal sequential write mode which is identical to the Vector Streaming Unit write ports (see 3.4.5). As each 512-bit group is taken by the port, the current store address is updated and sent, along with the data, to Main Memory. The second mode is used only for the Scatter operation. In this mode, the index for the address of the data to be stored is sent from the READ 3 port. The index is added to the base address before being sent to memory.

The data managment of the WRITE port is the same as the write ports in the other units - the left-most word of the 512-bit sword must be "right-adjusted" to put the data in the proper position to be stored into memory. Write enable flags are included with the data to show which of the 32-bit half-words hold valid data to be stored in memory. There can be invalid data on the first and last store request of an operation as well as invalid data on Scatter operations for shorter record lengths (for a single-word-record Scatter there might be only 32 bits valid in a 512-bit sword).

3.5.1.7 Operational Descriptions

3.5.1.7.1 Gather

A Gather operation proceeds as follows:

1) READ 3 will supply the indexes to READ 1 and READ 2. If the instruction specifies an index list in memory, READ 3 fetches the index data. If the instruction specifies a stride operation, then READ 3 modifies the specified strides so that the READ 1 and READ 2 ports can fetch alternate records. The indexes are then sent, two at a time, one each to READ 1 and READ 2.

2) READ 1 and READ 2 access memory, READ 1 first and then alternately. This is controlled by whose data is accepted into the ASSEMBLY element.

(continued)

3.5.1.7.1 (Cont.)

3) Records are assembled into 512-bit swords in the ASSEMBLY element. Data from READ 2 has passed through the COMPRESS element which is simply part of the data path.

4) Assembled swords are written into memory by the WRITE port.

3.5.1.7.2 Scatter

1) READ 3 will supply indexes to the WRITE port. If the instruction specifies an index list in memory, READ 3 fetches the index data. If the instruction specifies a stride operation, READ 3 computes the required indexes which are sent one at a time to the WRITE port.

2) READ 1 fetches the data, some of which may be held in the read port depending on the record size.

3) Data is sent to the ASSEMBLY element which performs no operation on the data but may hold it temporarily, depending on the record size.

4) 512-bit swords are sent to the WRITE port to be written into memory.

3.5.1.7.3 Compress

1) READ 3 fetches the control vector and sends it, 8 bits at a time, to the COMPRESS element.

2) READ 1 fetches data and sends it 8 operands at a time to the COMPRESS element.

3) The COMPRESS element sends the result data to the WRITE port through the ASSEMBLY element which is simply part of the data path.

4) The WRITE port sends the result data to memory.

3.5.1.7.4 Mask/Merge

1) READ 3 gets the control vector from memory and sends it, 8 bits at a time, to the MASK/MERGE element.

2) READ 1 fetches the first operand stream.

3) READ 2 fetches the second operand stream.

4) Both streams are operated on in the MASK/MERGE element under control of the control vector.

5) The resultant data stream is sent to memory by the WRITE port.

3.5.1.8 Operations with the Intermediate Map Unit

When operating with the Intermediate Map Unit, the required functions for a particular operation are shared between the two units.

If the index list for Gather and Scatter is a memory list, the list can be supplied from either Main Memory or Intermediate Memory.

On a Gather operation, the Intermediate Map Unit gets the data under control of the supplied index list and sends the data to the ASSEMBLY element of the Main Map Unit. The ASSEMBLY element operates identically as before as it assembles 512-bit swords to be stored.

On a Scatter operation to Intermediate Memory, READ 1 gets the data and sends it to the Intermediate Map Unit.

On a Compress operation, the data comes from the Intermediate Map Unit which sends it to the COMPRESS element to be operated upon. Of course, READ 1 and READ 2 are idle.

3.5.2 Intermediate Map Unit

The Intermediate Map Unit peforms the same functions for Intermediate Memory as the Main Map Unit performs for Main Memory. It is shown in block diagram form in figure 3.5-3. The functions of the individual elements of the Intermediate Map Unit are nearly identical to the corresponding functions of the Main Map Unit. Consequently, only the differences are discussed here.

The Intermediate Memory (discussed in Section 3.8) is a lower performance memory. It connects to the outside world through four 256-bit ports. The Intermediate Map Unit uses three of these ports.

The Intermediate Map Unit processes all its data on buses that are 256 bits wide unlike the Main Map Units 512-bit buses. Thus, the Intermediate Map Unit performs Compress, Mask, and Merge on four words at a time. When performing internal Gathers and Scatters (Intermediate Memory to Intermediate Memory, likewise, a maximum of 256-bit records are moved during one clock cycle.

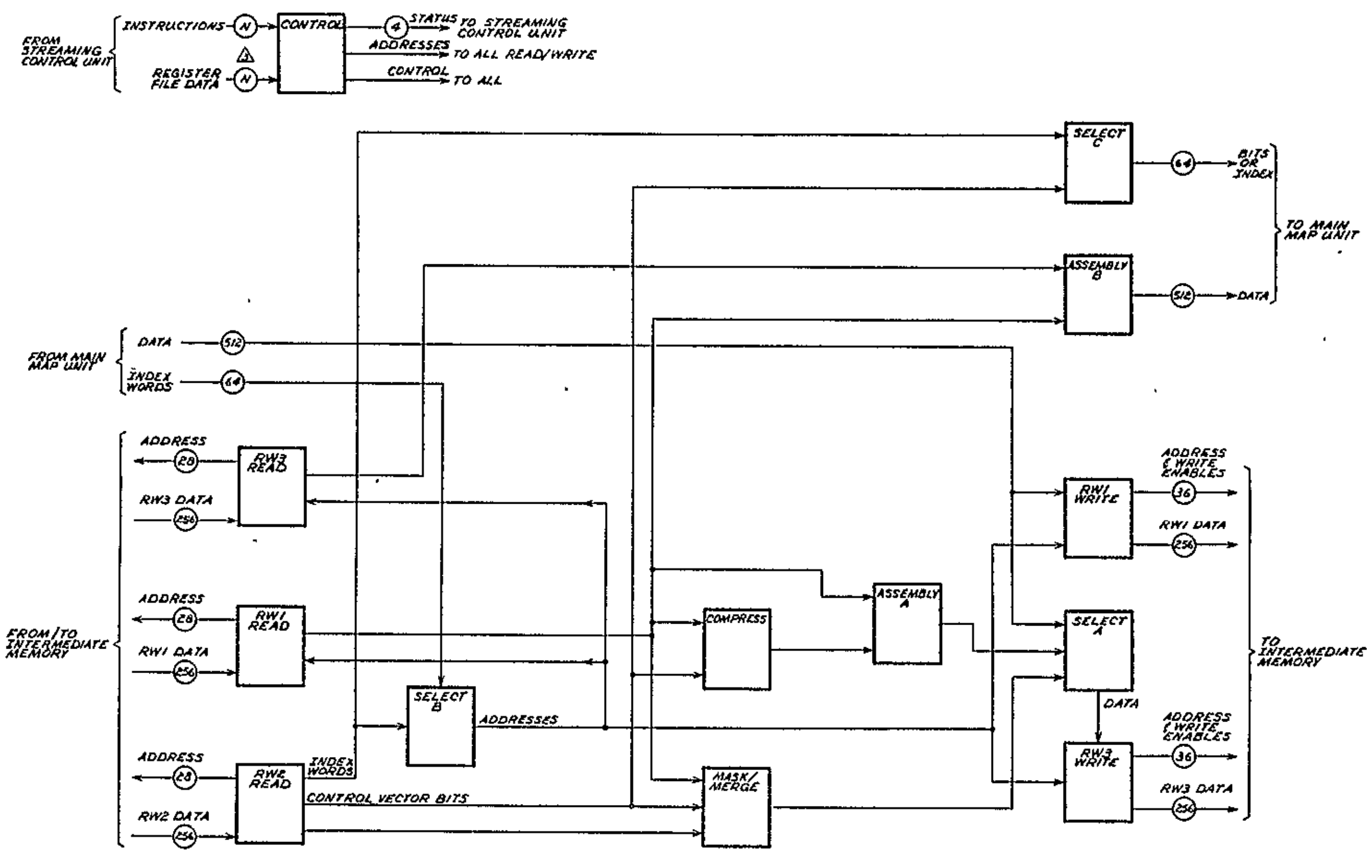
On Mask and Merge operations, four data streams are required:

1) A stream input,
2) B stream input,
3) control vector input,
4) result stream.

Because the unit has only three ports to Intermediate Map Unit, one of the ports must be shared between streams. Thus, the control vector stream and the B stream share port RW2. The loss in performance is relatively small because the control vector is a bit vector that is used only 4 bits every cycle, while 256 bits at a time are fetched. Thus, a new piece of control vector must be fetched once very 64 times (about 2%).

On a Gather operation in the Intermediate Map Unit, only one read stream is fetched from memory unlike the Main Map Unit where READ 1 and READ 2 alternate input records.

(continued)



NOTES
1. ALL DATA BUSES CARRY SECDED (NOT SHOWN IN BIT COUNTS).
2. RW"X" READ AND WRITE CANNOT BE USED SIMULTANEOUSLY
3. UNDETERMINED NUMBER OF BITS

Figure 3.5-3. Intermediate Map Unit

3.5.2 (Cont.)

For combined map unit operations, since data is being fetched from one memory and being stored in another, either the read or write data ports of a unit are not being used. The Intermediate Map Unit therefore, gets double use from a port by making what is normally either a read or a write port double purpose. This essentially doubles the rate for data transfers between Intermediate Memory and Main Memory of 256 bits (4 words or 8 half-words) every three clock cycles.

Thus, on a Gather or Compress operation from Intermediate to Main Memory, RW1 (normally a read port) and RW3 (normally a write port) both read data, 256 bits each, to be sent to the Main Map Unit and then to Main Memory. On a SCATTER, both RW1 and RW3 accept data to be written into Intermediate Memory. This feature doubles the bandwidth for intermemory transfers.

Intermediate Memory carries 8 bits of SECDED per 64 bits of data while Main Memory carries 7 bits of SECDED per 32 bits of data. When data is transferred between Map Units the SECDED information is transformed from one mode to the other.

3.6 Memory Interchange

The Memory Interchange (see figure 3.6-1) serves as the main memory access and control element. It coordinates all the memory read and write requests in such a manner as to assure the maximum throughput for Main Memory.

Memory Interchange gets requests for data from the following:

Read Requests	Number	Write Requests	Number
Vector Streaming	(4)	Vector Streaming	(2)
Main Map	(3)	Main Map	(1)
Scalar	(1)	Scalar	(1)

Each request is treated separately and is interfaced to a logic element called a port.

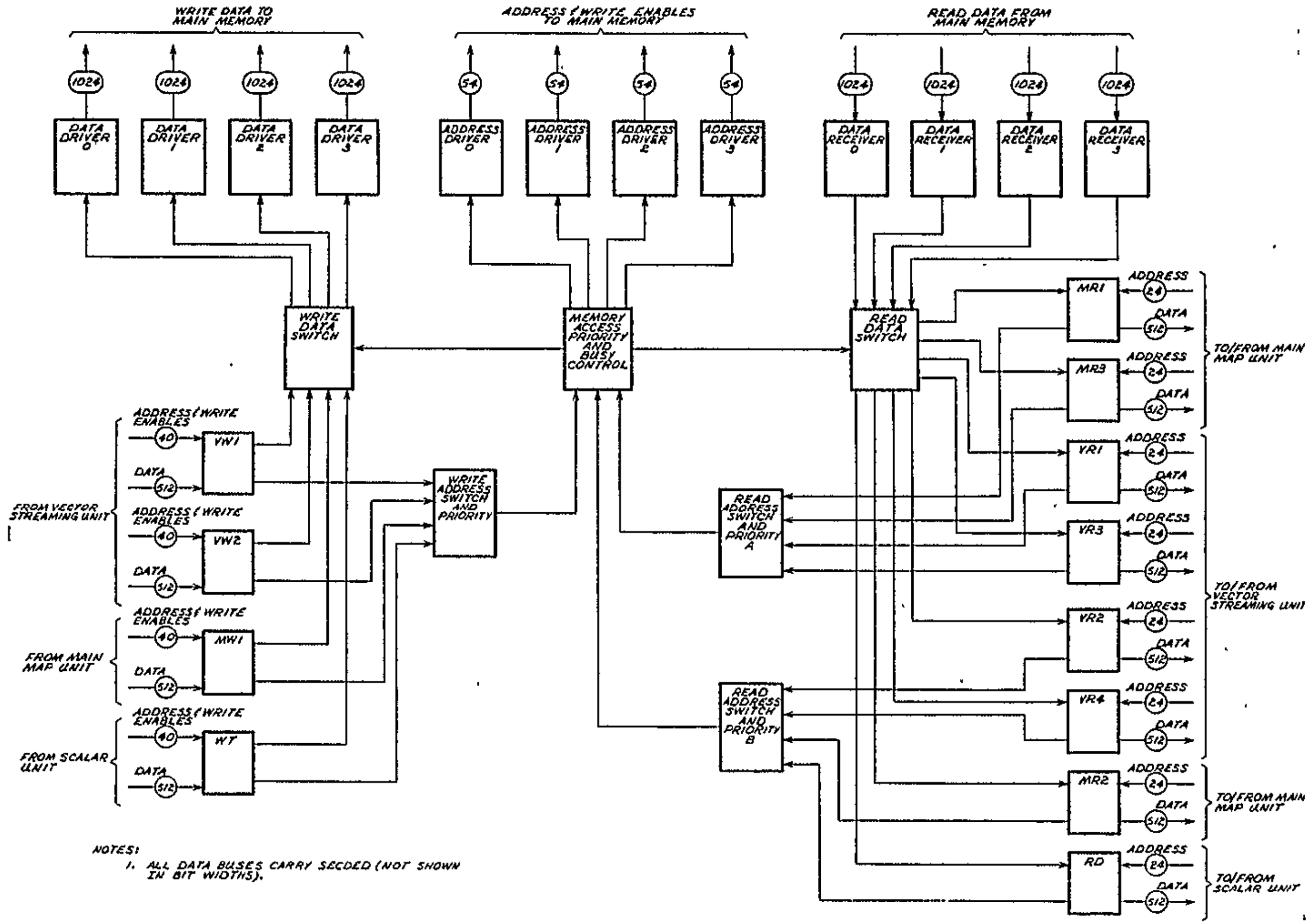


Figure 3.6-1. FMP Memory Interchange

3.6.1 Ports

3.6.1.1 Read Ports

Each read port carries 512 data bits plus 112 bits of SECDED information and connects to a single requestor. The read port performs the following functions:

1. Management of addresses and queuing of requests -- Because of the physical distances involved and the throughput requirements, a normal handshake protocol between the requestor and the port will not work well. The port thus has a queue that receives memory requests; it then makes a request of the memory priority logic and, when the priority logic grants access, sends back to the requestor an acknowledgement called "alert". The requestor keeps a count of the number of requests outstanding and this can avoid overflowing the address request queue.

2. Management of data -- Data, as requested from memory, is 1024 bits wide (+ SECDED) while all the data buses are 512 bits wide. When data is taken by the port it enters a queue and the requesting address is examined to see if the request is for the first or second half of the data. If the request is for the second half of the 1024 bits, the first half of the data is discarded.

If the port is in streaming mode (the vector read ports and the map unit ports when notified by the Map Unit) it attempts to stay ahead of requests. The port sends a flag to priority indicating that it is in streaming mode so that the priority unit will reserve memory access slots in advance for that port. If data then comes from memory faster than the requestor can take it, the data is accumulated in a data queue. If the queue becomes full the port drops the streaming request and will not resume the request until the queue is about half empty.

3.6.1.2 Write Ports

The write ports operate much like the read ports except that data is sent along with the memory address and request. As the data is accumulated the successive addresses are compared, or the port is notified of streaming mode, and this information is used to build up sets of 1024 bits that, together, will be written into Main Memory. Data write enables are kept with the data. The write enables, one with each 32 bits, declare the accompanying data valid. The write enables are sent, along with the data, to the memory.

3.6.2 Priority and Memory Control

Main Memory consists of four independent 1024-bit accesses. The Memory Interchange is accessed by 12 ports of 512 bits. (all data also carries SECDED). Because the requests are asynchronous with respect to one another and to independent addresses, the interchange control needs an extensive control network to maximize access to memory. The priority and control rules currently envisioned are as follows:

1. Memory access will be granted through 8 slots. Each slot can be granted access to memory every other cycle. (This is only a very limited restriction because the ports make two 512-bit swords into one 1024-bit access.)

2. A port, granted streaming access to memory, is assigned a slot and the slot is guaranteed to the port for the length of the streaming access.

3. Streaming requests have higher priority than non-streaming memory access requests.

4. Write requests have higher priority than read requests, primarily for two reasons. First, this will cause less conflict internally in the FMP functional units (data will get stored, will not backup). Second, there are considerably fewer writes than reads, which should allow the priority logic to be simpler.

(continued)

C-2

3.6.2 (Cont.)

Each bank of Main Memory is busy for three cycles when it is requested to read or write data. The control element keeps a register of 32 bits with one bit in the register for each seperately addressable bank in memory. Each bit is set by an access request that has been granted and is cleared automatically three cycles later. The bit register is referenced by the control unit as part of the access sequence to prevent access of already busy memory banks.

Along with the busy register is a reserved register with the same attributes as the busy register. This register is used by streaming requests to prevent non-streaming (random) requests from interfering with streaming operation. The bits of the reserved register are set two cycles ahead of setting the corresponding bit in the busy register. The reserved bit is checked along with busy bit when a non-streaming reference is made.

An access sequence for memory runs as follows:

1. A request arrives at a port. The request (ignoring data) consists of a memory address and a notification of streaming mode or not.

2. The request is sent to a first level priority network which can decide priority on up to four priority requests. The request address is broken down into 3 parts:

    a. The access requested,

    b. The bank number in the access,

    c. The address in the bank.

   The access numbers are compared against the other ports in the group. Any requests (possibly all four) that do not conflict are sent to the next level of control. Any conficts are resolved by applying the priority rules among the group. Any requests denied access are held until the next clock cycle.

3. The second level priority network checks each of the surviving first level requests against the busy register and also against the reserved register for non-streaming requests. Any requests still surviving are applied against the priority rules. The survivors are then gated to memory. The control element gates write data to memory for a write operation and puts the port number in a queue that is used by the read receivers to gate data to the proper port for a read operation.

3.7 Main Memory

Main Memory is a single-level, random-access memory using bipolar, 4K-bit integrated circuits. The memory words are 78 bits which provide for a 64-bit data word and 7 bits of single error correction double error detection (SECDED) for each 32-bit half-word. The semiconductor memory access time is 48 nanoseconds, where access time is defined as the time from the address reaching memory until data is clocked out of the memory. This memory is directly addressable in either monitor mode or job mode.

The basic Main Memory size is eight million words. The memory is designed so that it may be expanded in size at some future date.

Each two million words of Main Memory has its own separate 1024-bit interface called an access. Each access contains 16 memory modules each having 128K 78-bit words (64 data bits plus 14 SECDED bits). Each 128K module is arranged in eight phased banks. In streaming mode, a reference will be made simultaneously to the same address in each of the 16 memory modules to obtain a superword (sword) of 1024 data bits. Memory busy conflict rules take into account the 16 physically independent modules and the eight-bank phasing within each module to treat the bank address in each of the 16 modules as a separate entity. Thus, it could be said that each two million words of Main Memory contains 128 phased modules.

The eight-bank phasing plus the physical distribution of the memory modules allows memory references to be made at a maximum rate of one every minor cycle for each two million words of memory. Thus, the Main Memory has very high data transfer bandwidths:

one access = 1024 bits/minor cycle

two accesses = 2048 bits/minor cycle

four accesses = 4096 bits/minor cycle

3.7.1 Memory Module

The memory module is packed in a freon-cooled .5 cubic ft. volume with 8 banks, each bank containing 16K x 78 bits. There are three board types used in the module: input control, storage, and output. Figure 3.7-1 shows the module organization which lends itself to massive use of distributed loading and emitter-ANDing and also results in "zero-skew" construction which equalizes signal paths through all memory chips to maintain identical timing throughout the module.

185 coax lines connect each memory module to the Memory Interchange. All signals on the lines except the read data are sent from the interchange to the module. Below is a list of these lines:

Clock (2) - One for each input board to synchronize the memory module to the interchange.

Absolute Address (14) - Twelve address bits for the selection of the 4K memory chips and two address bits for the selection of the four ranks of memory chips.

Bank Address (6) - Three for each input board which are decoded for the selection of the eight banks within a module.

Module Request (2) - One for each input board which are decoded for selection of a unique memory module.

Write Control (2) - One write enable for each 39-bit half-word.

Write Data (78) - 78 data bits to memory, 64 for data, 14 for SECDED.

Sync (1) - This signal provides a point of time reference for maintenance purposes.

Master Clear (2) - One for each input board.

Read Data (78) - 78 Read data bits from the read data registers on the output board back to the interchange.

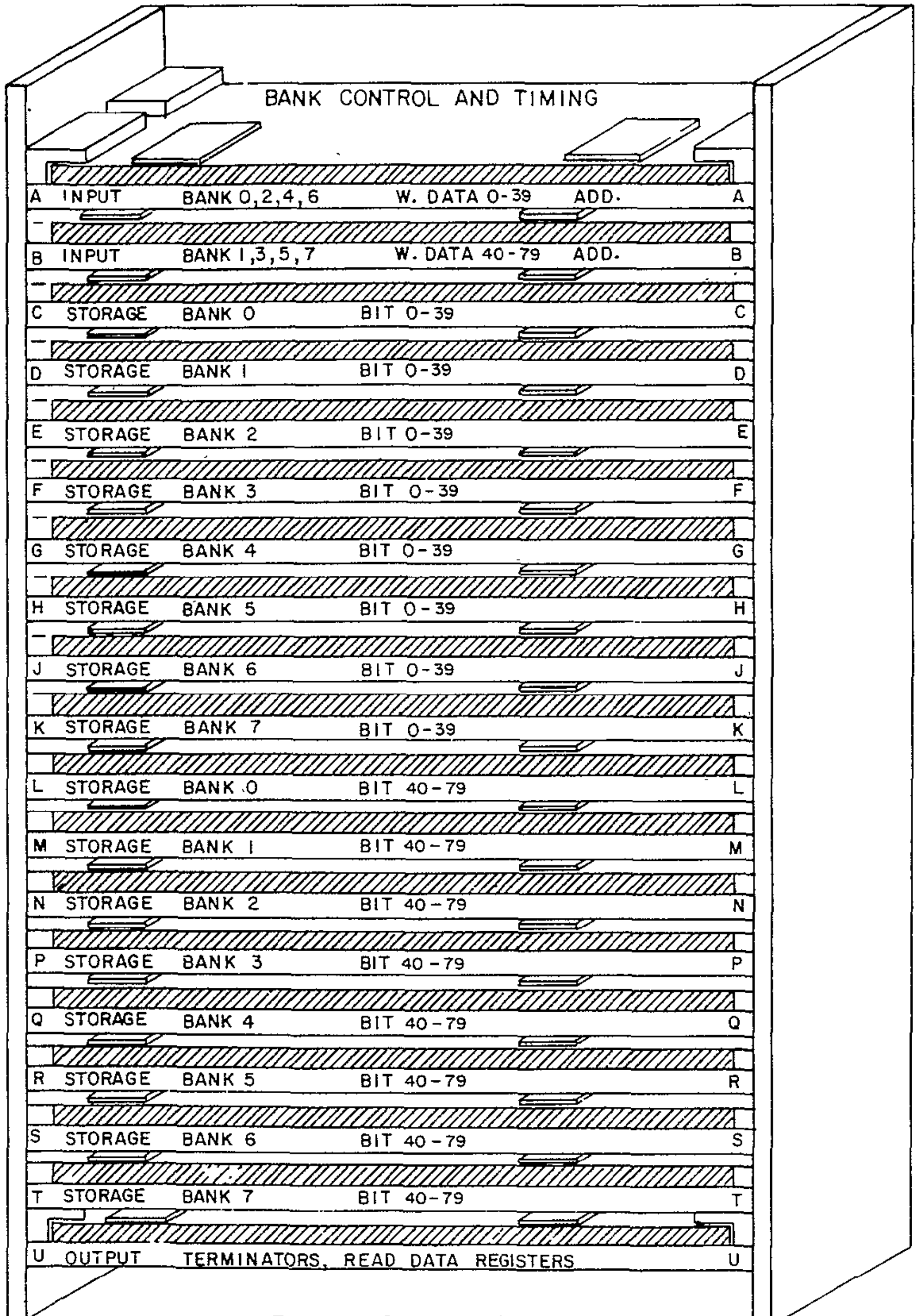


Figure 3.7-1 Memory Module

3.7.2 Memory Configuration

Memory modules are located within a section as shown in figure 3.7-2. There are eight modules per memory section and these sections are positioned as shown in figure 3.1-2. Each section contains one million words (64 data bits and 14 SECDED bits); thus, the eight million word memory is housed in eight sections.

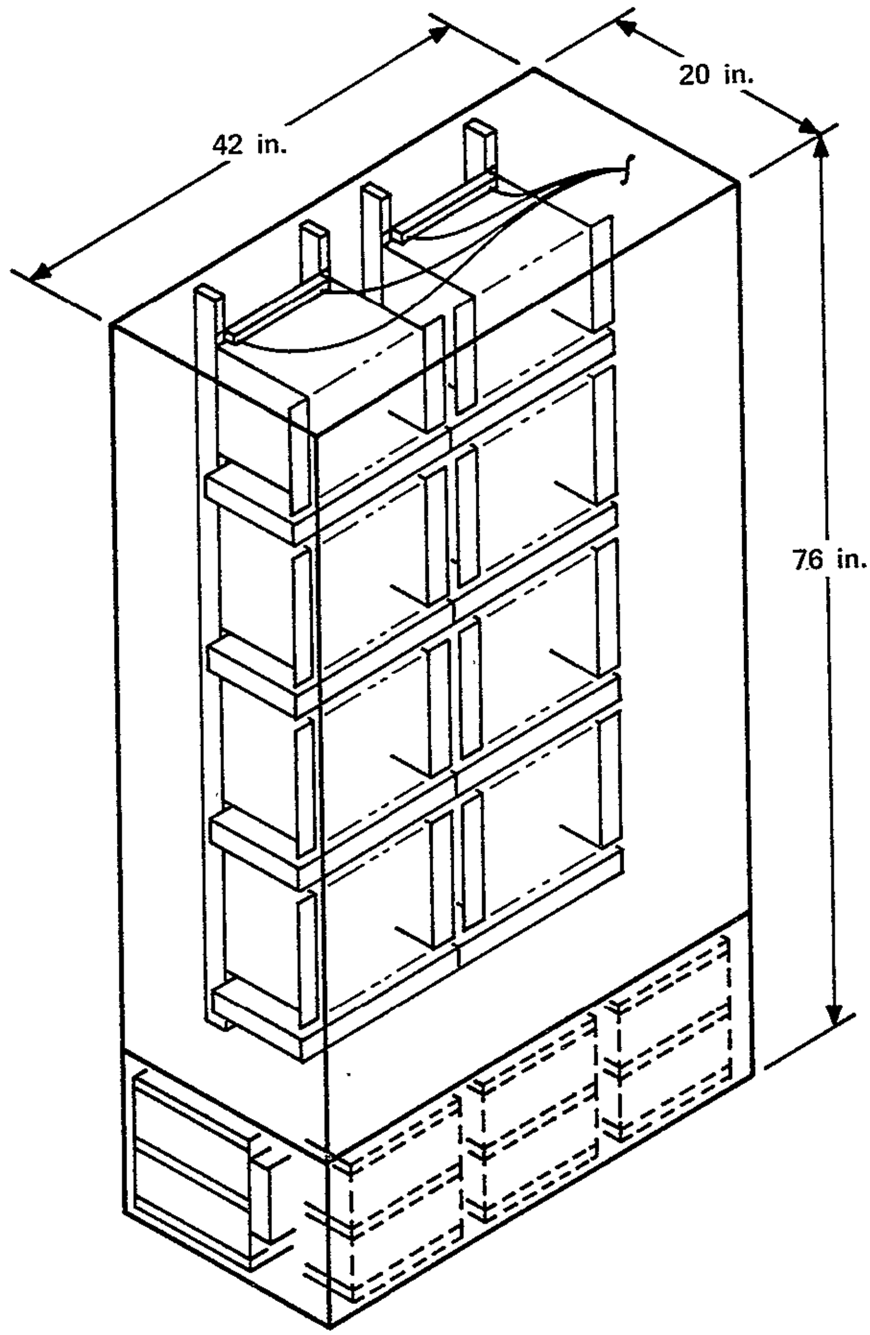


Figure 3.7-2 FMP Memory Section

3.8 Intermediate Memory

The Intermediate Memory consists of 33,554,432 64-bit words of random access memory. It is accessed through four high speed ports and up to eight low speed ports. The memory is shown in block diagram form in figure 3.8-1.

3.8.1 Organization and Access

The Intermediate Memory is organized as four memory groups with each group having four memory banks and each bank having eight memory modules. Each module has 262,144 72-bit words (64 data bits plus eight SECDED bits). The four memory banks in each group are driven in parallel; that is, the same address request is sent to each bank in the group at the same time. Thus, data is available at the output of the memory group 288 bits wide (256 data plus 32 SECDED).

When a memory group is accessed, the memory control interprets the lower three bits of the starting address to determine an initial module number. That module set (four modules in parallel) will be accessed, and every 48 nanoseconds thereafter the next successive module set will be accessed through the remainder of the 32-word block. Thus, a request to a group will get a variable amount of data. If the three bits of the request address are 000, then eight 288-bit data transfers will result. If the lower bits of a request address are 101, then three 288-bit transfers will result (module groups 5, 6, 7). The cycle time of a memory group is 384 nanoseconds (24 FMP clock cycles).

Because of the anomaly caused by the modules of a group not starting simultaneously, throughput to memory is maximized by starting requests at module set 0 (lower address bits 000) as much as possible, and by making as much use as possible of the 32 words thus transferred. The port controls of the Map, I/O, and Swap Units have this built into them.

3.8.2 High Speed Ports

Each of the four high speed ports can move data at a rate of 288 bits every 48 nanoseconds. Each port has a small buffer to hold up to 32 words in case the requested memory group is busy or the port is denied access because of priority.

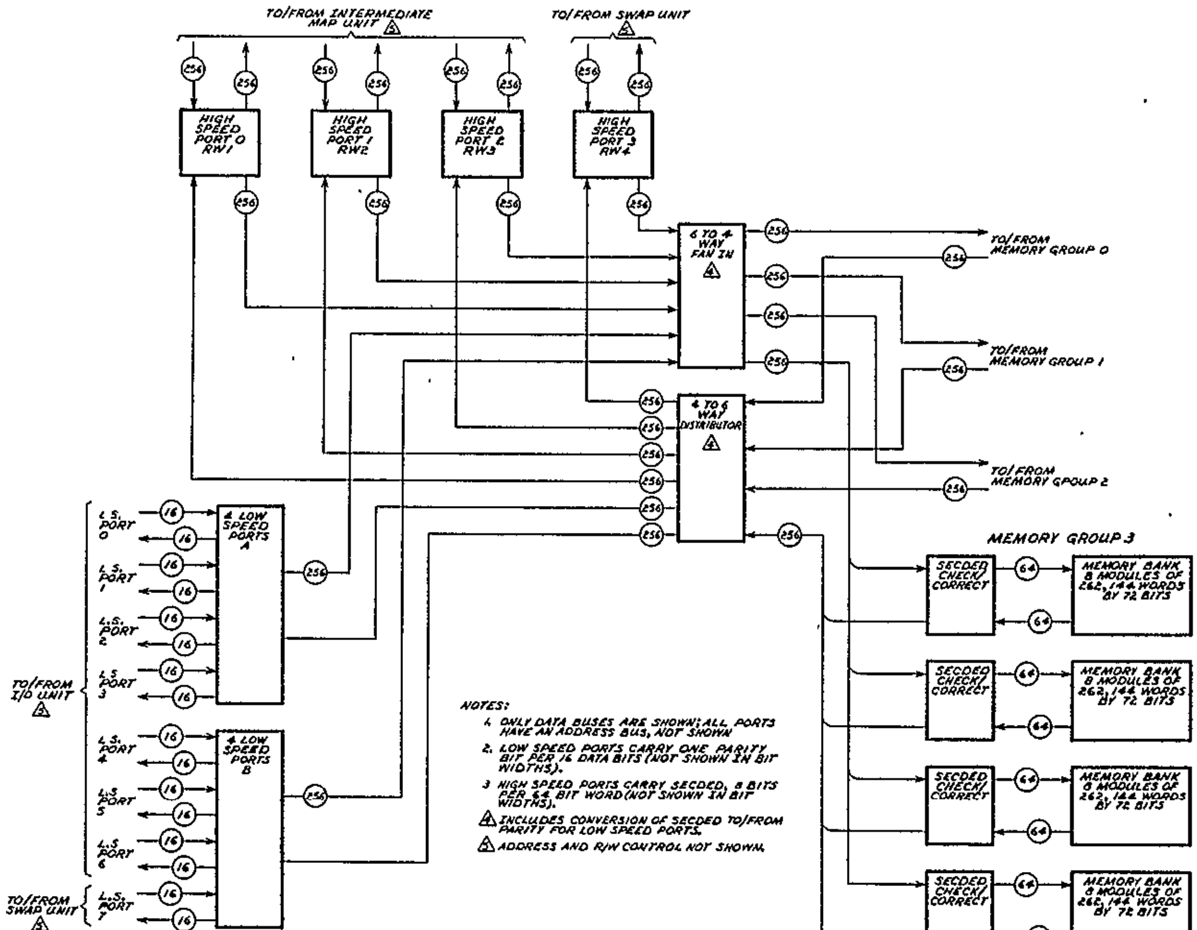


Figure 3.8-1 Intermediate Memory

3.8.3 Low Speed Ports

The low speed ports are made into two sets of four ports. Each set appears to the memory control as a high speed port. Thus to memory control, the memory is accessed by six high speed ports. Each low speed port can move data into and out of the port at a rate of 17 bits (16 data plus one parity) every 96 nanoseconds. Each low speed port also has a 32-word, 64-bit buffer. If a write operation to Intermediate Memory does not terminate on a 64-bit boundary the rest of the 64-bit word is zero filled when the data is written. For this reason the I/O Unit and Swap Unit perform a read-modify-write for a write operation which does not terminate on a 64-bit boundary, thereby avoiding the zero fill.

3.9 Input/Output

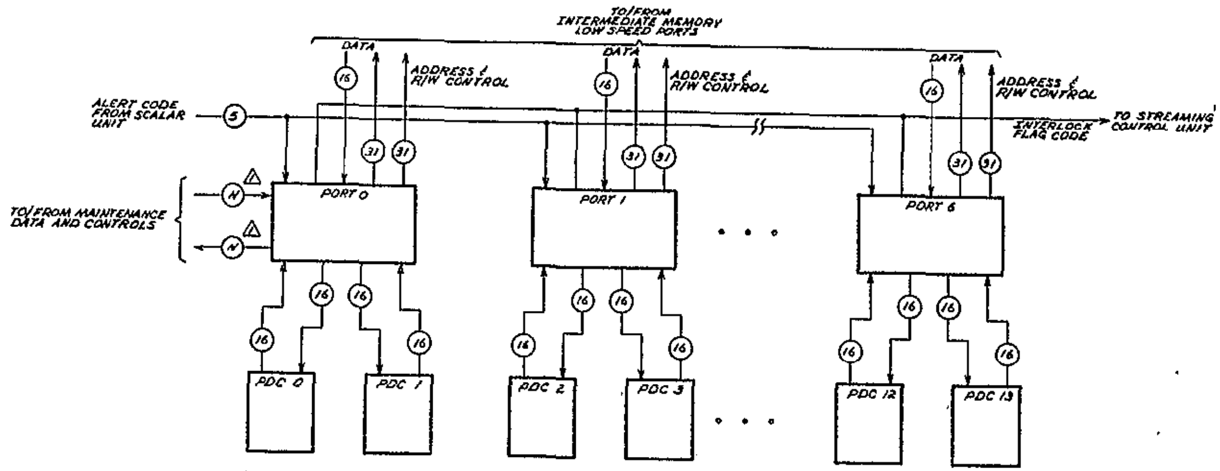
FMP I/O consists of all the functional elements of the processor that are normally considered external to a user's running job. These elements are:

1) the I/O Unit which consists of a variable number of input/output processors (PDCs) connecting the FMP to the total system via the loosely coupled network (LCN). All I/O data move from/to the LCN to/from the Intermediate Memory (described in section 3.8);

2) the Swap Unit which connects the Backing Store to Intermediate Memory. The 128 million-word Backing Store (see section 3.10) is used by the system and by the user job to perform local high speed I/O. All user job READ and WRITE statements refer to the Backing Store where all output (print) files are retained until a job is finished; they are then passed to their final destination.

3.9.1 I/O Unit

The I/O Unit moves 32K-word data blocks as well as system messages to and from Intermediate Memory. Each I/O channel consists of a PDC processor as well as hardware to connect it to Intermediate Memory. Each connection to Intermediate Memory (called a port) is capable of being shared with another PDC processor. There may be from 1 to 7 ports and thus from 1 to 14 I/O channels. The I/O Unit is shown in figure 3.9-1.



NOTES:
1. UNKNOWN NUMBER OF BITS.
2. PARITY CARRIED ON ALL 16 BIT DATA LINES (NOT SHOWN).
3. ONLY PDC'S 0 AND 1 CAN FUNCTION AS THE MCU INTERFACE.
4. THERE MAY BE A MAXIMUM OF 7 PORTS AND 14 PDC'S. (THE BASIC SYSTEM WILL HAVE 3 PORTS AND 6 PDC'S).
5. NO DROPS TO THE SERIAL TRUNK SYSTEM ARE SHOWN BECAUSE THE NUMBER OF DROPS FROM EACH PDC IS DETERMINED BY THE SYSTEM CONFIGURATION AND RELIABILITY REQUIREMENTS. (4 DROPS/PDC MAXIMUM).

Figure 3.9-1 I/O Unit

3.9.1.1 The PDC

Each PDC (described fully in appendix C) consists of 4 parts.

1) The frontend provides connection of the PDC to the serial trunk network. The frontend may attach from 1 to 4 trunks. On reads the frontend verifies that a message or data block on the trunks is destined for this PDC, or it ignores the data. When the frontend reads data from the trunk, it converts the data to 16-bit words and stores the data into the PDC buffer. On write sequences, the frontend serializes the data and puts it on the proper serial trunk.

2) A data buffer with 16-bit data width is a repository for data on its way through the PDC, either front to back or vice-versa. The buffer also holds code for execution by the PDC processor. The buffer size may be from 8K to 32K words or more, depending on requirements.

3) The backend consists of logic to connect to the FMP port. It performs handshaking and control to move data between the I/O port and the data buffer.

4) The processor is a 16-bit, bit-sliced, bipolar, microcoded computer made specifically for its communication function.

   The processor is in control of both the frontend and backend and is notified by them upon completion of any function, and also of any exception conditions. The software executed by the processor is normally "downloaded" from the Maintenance Control Unit when the system is autoloaded. Because the processor is intelligent, it cannot only manage data on its way through the PDC, but can transfer the data under software control, changing character sets or file formats, for example.

All PDCs are the same throughout the LCN system. The only thing that differentiates each is its "backend" which matches the particular device to which the PDC is tied. Each PDC may, of course, run different software.

(continued)

3.9.1.1 (Cont.)

When an FMP I/O PDC is running it waits for a request to perform a function that arrives from either end. Upon receipt of an ALERT code from the backend (ALERT is sent by the Scalar Unit to a particular I/O PDC) the PDC processor functions the backend to read a message block from Intermediate Memory. The processor scans the message, determines the required function, and performs it. At the termination of the function a completion message is left in Intermediate Memory and a signal flag sent to the Scalar Unit, notifying it of the message.

When a PDC receives a data block from the trunk system, it puts the data into a preassigned area of Intermediate Memory.

All data sent on the trunk system carries a cyclic redundancy check code.

Data messages sent on the trunks generally consist of 2 parts, a header and the data. Each part has a separate redundancy check so that a data error is separate from a function (header) error. On a header error, no reply is sent to a received message because it could be the destination file of the header that is in error. If the header redundancy check is okay but the data check is bad, then a reply is sent to the originating PDC telling of the data error. The sending PDC will then retransmit the data block (if the PDC software so indicates).

3.9.1.2 I/O Ports

The FMP will accommodate, optionally, from 1 to 7 I/O ports. Each port can support one or two PDCs. The principle function of the port is to manage data going to/from Intermediate Memory and the attached PDCs. See figure 3.9-2 for block diagram of the I/O port. Intermediate Memory can transmit 16 bits (plus parity) of data every 96 ns to the port. The PDC can give or take 16 bits (plus parity) every 320 ns. The port provides a FIFO buffer for each attached PDC; the buffer will accumulate, on a write to Intermediate Memory, 32 64-bit words and then the port control element will automatically request Intermediate Memory to accept the data. While the data is being transferred to Intermediate Memory, the PDC can keep transferring data to the buffer. On a read from Intermediate Memory the process is reversed, the port requesting 32 64-bit words of data, notifying

(continued)

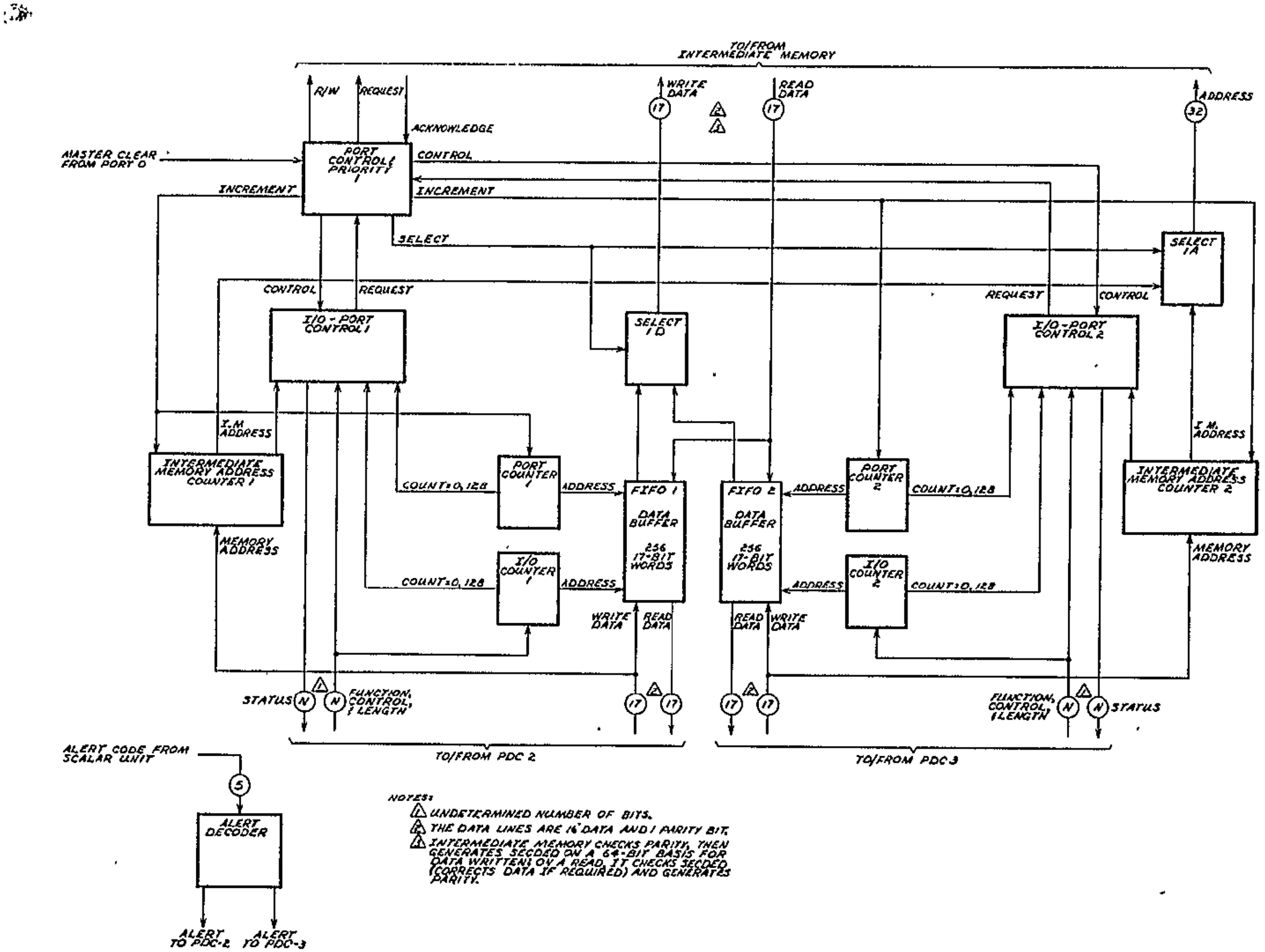


Figure 3.9-2 Port 1 of I/O Unit

3.9.1.2 (Cont.)

the PDC of the data availability, and automatically making requests to Intermediate Memory to stay ahead of the reading PDC. The port is sent an initial Intermediate Memory address by the PDC before any data transfers are started. The port increments the address automatically as each memory request is made. the port also contains priority logic to decide a memory access sequence if the two PDCs make memory requests at the same time.

A small difficulty arises because the PDCs are 16-bit machines while Intermediate Memory is based on 64-bit words and only writes whole 64-bit words, even though a PDC does not have to send an integer number of 16-bit words. For the case where the last 64-bit data word is not filled by the PDC, Intermediate Memory will zero fill the remainder of the written word. In order to prevent this from happening, if the port is in a write sequence and receives a termination signal from a PDC without a complete 64-bit word in the FIFO buffer, the port will automatically fetch the odd word, combine it with the partial word in the buffer, and then write the last data to Intermediate Memory.

3.9.2 Swap Unit

The Swap Unit has two main functions: to move data between Intermediate Memory and the Backing Store; to provide an interface for the Scalar Unit to have access to Intermediate Memory.

The Swap Unit shown in figure 3.9-3 is connected to high speed port 3 and to low speed port 7 of Intermediate Memory. The high speed port is used to transfer backing store data and also scalar unit data. The low speed port is used by backing store control to receive command messages from the Scalar Unit that have been left in Intermediate Memory (and then to return status replies).

The high speed port can provide/take data at the rate of 256 bits plus SECDED every 48 ns. The Backing Store, as presently envisioned, will move data at the rate of 512 bits every 256 ns -- equivalent to a rate of 256 bits every 128 ns. Thus Backing Store uses 3/8 of the port bandwidth. The rest of the bandwidth is available to the Scalar Unit.

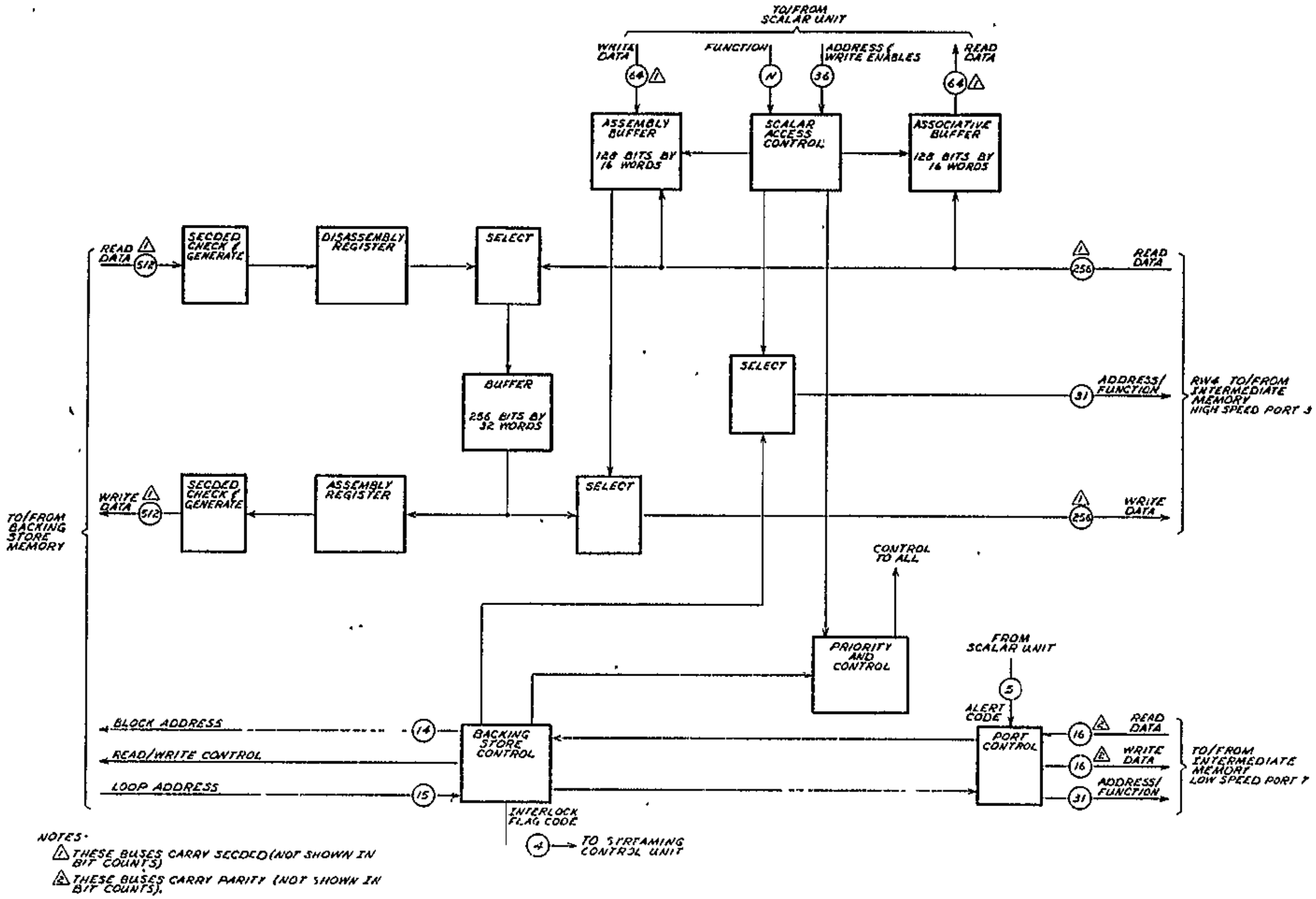


Figure 3.9-3 Swap Unit

3.9.2.1 Scalar Interface

The connections to Intermediate Memory are 256 bits wide but the Scalar Unit does 32-bit and 64-bit reads and writes. In order to reduce conflicts, the scalar interface has two small buffers, 32 64-bit words each, for read and write. On a read the port will get a 32 word block that contains the requested data word, holding it the read buffer. On subsequent read requests, if the requested word is in the buffer, the interface will return the word from the buffer without making an Intermediate Memory request.

On a write, the write buffer will attempt to build several store requests from the Scalar Unit into a 32-word group (with some holes likely) before making a memory request. Any request that does not fit in the present 32-word buffer (because it is outside the address range of the 32-word group) will cause the buffer to be written to Intermediate Memory and a new buffer data set started. Any write address that also exists in the read buffer will void the read buffer.

3.9.2.2 Backing Store Interface

Data is moved to/from the Backing Store in blocks of 32,768 64-bit words, the same size as data blocks in the rest of the system.

The swap control connection to Intermediate Memory is identical to that of an I/O port. To the FMP, the Swap Unit appears to be an I/O port. Thus the command sequence to move data to and from the Backing Store are requests to a disk for data - cylinder and sector selects become addresses in the Backing Store. The difference, of course, is that the data moves on a separate high speed link to/from Intermediate Memory.

The Backing Store is a block organized memory. As such, it would suffer an access time to the first data in the block which would appear to the system as a 33% reduction in throughput.

In order to avoid this access time, the Backing Store maintains a counter that points to the current address within a block that can be fetched immediately with no time lost. When the Swap Unit is given a command to move a block it requests the current

(continued)

3.9.2.2 (Cont.)

count and uses that count to modify the address to be requested first in Intermediate Memory. Data transfer thus may start anywhere in the data block. When the end of the block is reached, swap control resets the intermediate memory address to the first address in the block and continues to transfer data. When the original transfer address is reached, the block transfer is complete.

A buffer is provided in the data path between the two memories which serves two purposes:

1) As a speed matching device between the two memories, the buffer will access Intermediate Memory in 32-word groups of 64-bit (plus SECDED) words at a rate of 4 words per 48 ns.

2) To allow the scalar unit interface higher priority for intermediate memory accesses, the buffer can hold up to 4 32-word groups. If the scalar requests should hold out backing store requests for a time such that the buffer is in danger of overflowing (Backing Store -> Intermediate Memory) or of becoming empty (Intermediate Memory -> Backing Store), the access priority is switched giving buffer access request priority.

3.10 Backing Store

The Backing Store is 128 million 64-bit words of memory built from charge coupled devices (CCDs) which contain either 65K bits/chip (available today) or 256K bits/chip (samples expected late 1979). Using the 65K device each board in the memory system will hold 1 bit. There will be 576 memory boards (512 data + 64 SECDED). Because of SECDED a whole board can fail and be covered by SECDED. For a system using 256K chips, there will be a total of 144 boards, each with 4 bits. Complete board coverage can still be obtained by placing each bit in a separate syndrome group. Thus a complete board failure will cause 4 separate single-bit failures, each corrected.

Data is moved in the Backing Store 512 data bits at a time or 8 64-bit words. Within each CCD chip is a set of data loops 4096 bits wide. The effect of moving 8 words at a time is to make each loop appear externally as 32,768 bits long. This then sets the block size for the machine.

3.11 Maintenance Control Unit (MCU)

The MCU is an autonomous Maintenance Control Unit connected to the computer via a CDC FMP I/O channel which has access to special internal interfaces. These interfaces allow it to regulate information flow and control and to monitor performance of the computer. The MCU consists of a control unit, line printer, card reader, and a disc drive in a stand-alone configuration; through its connection to the I/O channel it provides for system dead-start and system performance monitoring. Diagnostics and preventive maintenance are run and controlled by the MCU.

There are two operating modes for the MCU.

1. The first mode of operation is running diagnostic routines on the FMP. The MCU loads diagnostics, ranging from a simple command test to a very sophisticated diagnostic catalog routine, controls and monitors the operations of the diagnostics, and displays the results of the tests via the display unit or line printer.

2. The second mode of operation is system operation. The MCU loads the Operating System software into the FMP and controls and monitors its operation. In this on-line mode of operation, the MCU concerns itself with autoloading the central processor and first level stations, running on-line diagnostics, monitoring CPU faults, and restarting the central processor after hang-ups. The MCU can also be used to monitor system or job performance.

3.11.1 MCU/CPU Interface

The MCU connects to the FMP via the loosely coupled network trunk system. The maintenance interface, contained in the I/O Unit, has 16 buffers called MCU/CPU channels - ATB being outgoing buffers and BTA being access channels. Tables 3.11-1 through 3.11-8 show the channels from the CPU to the MCU (ATB) and Tables 3.11-9 through 3.11-16 show the channels from the MCU to the CPU (BTA). Each table shows the channel bit number, and function of each bit for a channel.

3.11.1.1 Channels from CPU to MCU

TABLE 3.11-1 CHANNEL ATB1

Bit No.		Function
0	Bit 0	Current Instruction Address
1	1	Register
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
A	10	
B	11	
C	12	
D	13	
E	14	
F	15	

TABLE 3.11-2 CHANNEL ATB2

Bit No.		Function
0	Bit16	Current Instruction Address
1	17	Register
2	18	
3	19	
4	20	
5	21	
6	22	
7	23	
8	24	
9	25	
A	26	
B	27	
C	28	
D	29	
E	30	
F	31	

(continued)

3.11.1.1 (Cont.)

TABLE 3.11-3 CHANNEL ATB3

Bit No.	Function	
0	Bit32	Current Instruction Address
1	33	Register
2	34	
3	35	
4	36	
5	37	
6	38	
7	39	
8	40	
9	41	
A	42	
B	43	
C	44	
D	45	
E	46	
F	47	

TABLE 3.11-4 CHANNEL ATB4

Bit No.	Function	
0	Bit 0	
1	1	Display Register - Displays the
2	2	register selected by bits C-F of
3	3	channel BTA1 in the MCU.
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
A	10	
B	11	
C	12	
D	13	
E	14	
F	15	

(continued)

3.11.1.1 (Cont.)

TABLE 3.11-5 CHANNEL ATB5

Bit No.		Function
0	Bit 16	Display Register - Displays the register selected by bits C-F of Channel BTA1 in the MCU.
1	17	
2	18	
3	19	
4	20	
5	21	
6	22	
7	23	
8	24	
9	25	
A	26	
B	27	
C	28	
D	29	
E	30	
F	31	

TABLE 3.11-6 CHANNEL ATB6

Bit No.		Function
0	Bit 32	Display Register
1	33	
2	34	
3	35	
4	36	
5	37	
6	38	
7	39	
8	40	
9	41	
A	42	
B	43	
C	44	
D	45	
E	46	
F	47	

(continued)

3.11.1.1 (Cont.)

TABLE 3.11-7 CHANNEL ATB7

Bit No.	Function
0	Bit 48 Display Register
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
A	58
B	59
C	60
D	61
E	62
F	63

TABLE 3.11-8 CHANNEL ATB8

Bit No.	Function
0*	Memory SECDED Fault or Instruction Stack Parity
1*	Microcode Parity Fault
2	Not Used
3*	Absolute Sword Bounds Hit
4*	Event Stop
5*	Single SECDED Error
6	CPU Clock - Used for gating data back to the CPU. The MCU cannot read this line.
7	Monitor Mode
8	Temperature - Dew Point Alarm
9	Not Used
A	Section Power Fail
B	60 Hz Input Power Fail, M.G.1
C	60 Hz Input Power Fail, M.G.2
D	Not Used
E	CPU Idle
F	CPU Stopped

* These lines indicate why the CPU has stopped.

3.11.1.2 Channels from MCU to CPU

TABLE 3.11-9 CHANNEL BTA1

Bit No.	Function
0	MAC Master Clear - Master Clear to Memory Interchange, and Main Memory only. This includes the I/O channels. This signal must be set a minimum of 3 microseconds.
1	Stop - CPU will stop before next instruction issue.
2 *	Step - Execute one instruction. Store the register file and the invisible package (job mode only); then stop. Faults must be cleared before the computer can be stepped.
3 *	Run - Start CPU from manual stop or fault stop. Faults must be cleared before computer can be started.
4 *	Store Register File - The Register File is stored starting at address 0000_{16} in monitor mode and address 4000_{16} in job mode.
5 *	Load Register File - The Register File is loaded starting at address 0000_{16} in monitor mode and address 4000_{16} in job mode.
* Computer must be stopped before executing these commands.	

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-9 CHANNEL BTA1 (Cont.)

Bit	Function
6	CPU Master Clear - Master Clear to Scalar Unit, Stream, and Floating Point only. Memory Interchange, I/O Channels, and Main Memory are not included. This signal must be set a minimum of 3 microseconds.
7	Clear Fault Conditions - This signal clears the following conditions and allows the computer to be restarted with a run signal (bit 3): a. SECDED Double Error Condition b. MIC Memory Parity Fault c. Sword Bounds Hit d. The Bounds Hit Address is released. e. Reference to Illegal Address in Stream Microcode. f. Instructional Stack Parity Error
8	Clear SECDED Single Error, SECDED Fault Address and Syndrome Bits.
9	MCU Sync.- This signal is used in the CPU to gate the CPU data back to the MCU. When reading the display registers, the MCU Sync. signal must be set after the read signal is set.
A	Select SECDED Error Mode Two.
B	Read - Transfer selected register and CIAR into the Display Registers.
C D E F	Display Register Selection See Section 3.11.4.2

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-10 CHANNEL BTA2

Bit		Function
0 *	Static	Latch Memory Size Code
1 *		Interrupt Gate - When this signal is a "1", time interrupts and external interrupts will only be processed between instructions.
2 * 3 * 4 *		Memory Size Degrade Code 000 = 2 Meg Memory 001 = 2 Meg Memory, Force Section 1 --> Section 0 010 = 2 Meg Memory, Force Section 2 --> Section 0 011 = 2 Meg Memory, Force Section 3 --> Section 0 100 = 4 Meg Memory 101 = 4 Meg Memory, Force Upper 4 Meg --> Lower 4 Meg 110 = 8 Meg Memory
5 * 6 * 7 *		Select Mainframe Clock Freq. 000 = Nominal 001 = Increase clock freq. (1) 010 = Decrease clock freq. (1) 011 = Select variable freq. (adjustment on oscillator pak) 100 = Increase clock freq. (2) 101 = Increase clock freq. (3) 110 = Decrease clock freq. (2) 111 = Decrease clock freq. (3) NOTE: If clock frequency codes 4-7 are used, then code 3 is not available. Either codes 0-3 or 0-2 and 4-7 are available.
8 *		Delay Trailing Edge - Delay the trailing edge of all of the clocks on the panel which is specified by bits 11-15 of Channel BTA2. If bit 8 and bit 9 are set, only the odd or even clock, on a panel are moved depending on bit A.

* Computer must be stopped before executing these commands.

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-10 CHANNEL BTA2 (Cont.)

Bit		Function
9 *	Static	Delay Leading Edge - Delay the leading edge of all of the clocks on the panel which is specified by bits B-F of Channel BTA2. If bits 8 and 9 are set, only the odd or even clocks on a panel are moved depending on bit A.
A *	Static	"0" - Move even clocks(see description for bit 8 or 9). "1" - Move odd clocks.

* Computer must be stopped before executing these commands.

Bit	Function	
B (2^4) C (2^3) D (2^2) E (2^1) F (2^0)	Panel Designator for Clock Margins - Bit B is the left-most bit of the designator. The designators are defined below.	
	Designator$_{16}$	Panel(s)$_C$
	00	
	01	
	02	
	03	
	04	
	05	
	06	(to be supplied later)
	07	
	08	
	09	
	0A	
	0B	
	0C	
	0D	
	0E	
	0F	

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-10 CHANNEL BTA2 (Cont.)

Bit	Function
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F	(to be supplied later)

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-11 CHANNEL BTA3

Bit No.	Function
0	Not Used.
1	Send an external flag on the channel specified by the Channel Select Code in bits 4-8. (*1)(*2)
2	Set Channel Disable on the channel specified by the Channel Select Code in bits 4-8.(*1)(*3)
3	Clear Channel Disable on the channel specified by the Channel Select Code in bits 4-8.(*1)(*3)
4, 5, 6, 7	Channel Select Code. A code of 0_{16} thru D_{16} selects a channel (0_{10} thru 13_{10}) and F_{16} selects the Swap Unit for the operation specified in bits 1, 2 and 3.(*1) Bit 7 of BTA3 is bit 3 of the Channel Select Code.
8	Select All Channels (0_{10} thru 13_{10}) and the Swap Unit for the operation specified in bits 1, 2 and 3.(*1)
9	Stop on SECDED Single Error Detection.
A	Disable Stop on SECDED Double Error Detection.
B	Block External Interrupt
C	Disable Error Correction on all Read Buses.
D	Swap Register File Read on Exchange.
E	Not Used
F	Not Used.

(continued)

3.11.1.2 (Cont.)

(*1) The Channel Select Code bits 4-8 must be set before any commands are sent, and it must remain set until after the command has dropped.

(*2) The External Flag is transmitted to the device on the I/O channel corresponding to the code in bits 4-8. External Flag instructs the device to autoload.

(*3) The Channel Disables are transmitted to the I/O Unit. If the disable line for a channel is set, no intermediate memory references will be allowed from that channel. Data transfers can proceed in and out of the channel buffer in an end-around type of operation.

TABLE 3.11-12 CHANNEL BTA4

Bit	Function
0	Checkword bit 0 — Used for toggling I/O Checkword bits 0-6
1	Checkword bit 1
2	Checkword bit 2
3	Checkword bit 3
4	Checkword bit 4
5	Checkword bit 5
6	Checkword bit 6
7	Block Write Enable on SECDED Error
8	Not Used
9	Not Used
A	Force Register File Store at bit address 20000_{16} on Initial Exchange
B	Force Instruction Stack Parity
C	Enable I/O Simulator
D	Initiate I/O Simulator on Channel Flag
E	Not Used
F	Not Used

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-13 CHANNEL BTA5

Bit	Function
0-4	Not Used
5-7	Bounds Limit Load Code 0 = Null 1 = Load Bits (35-42) Upper Bounds 2 = Load Bits (51-58) Upper Bounds 3 = Load Bits (43-50) Upper Bounds 4 = Null 5 = Load Bits (51-58) Lower Bounds 6 = Load Bits (35-42) Lower Bounds 7 = Load Bits (43-50) Lower Bounds
8-F	Bounds Address Bits Due to the operational characteristics of the maintenance interface, only one bit of the code can be changed at one time. Address bits must be loaded in such a manner as to leave the Load Code bits undisturbed. Address bits are transferred on the Leading Edge of a code change, the address bits must be set up before a code change occurs. Address bits are Loaded as follows, starting and ending with a Null Code: Code = 0 Null Code = 1 Set up Bits (35-42) Upper Bounds Code = 3 Set up Bits (43-50) Upper Bounds Code = 2 Set up Bits (51-48) Upper Bounds Code = 6 Set up Bits (35-42) Lower Bounds Code = 7 Set up Bits (43-50) Lower Bounds Code = 5 Set up Bits (51-58) Lower Bounds Code = 4 Null Bounds limits are absolute, physical half-word addresses. Bits (35-36) and (55-58) must be zero.

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-14 CHANNEL BTA6

Bit	Function	
0	Check bounds on memory reads	If bits 0 and 1 or bits 2 and 3 are zero, no bounds hits can occur.
1	Check bounds on memory writes	
2	Check bounds on CPU references	
3	Check bounds on channel references	
4	Stop CPU on bounds hit	
5	Enable bounds check - The bounds addresses and conditions must be set up before the enable is set.	
6	Count A - Monitoring Counter A is enabled while this line is a "1" and held clear when this line is a "0". The proper counter specification and bits 8-E of channel BTA6 must not change while this line is up.	
7	Count B - Monitoring Counter B is enabled while this line is a "1" and held clear when this line is a "0". The proper counter specification and bits 8-E of channel BTA6 must not change while this line is up.	
8	Clear counter (see code 6 in Section 3.11.4.2).	
9	Stop CPU on Counter A Increment	See Section 3.11.4.1.3
A	Stop CPU on Counter B Increment	

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-14 CHANNEL BTA6 (Cont.)

Bit	Function	
B	Enable Carry into A1	See Section 3.11.4.1.2
C	Enable Carry into A2	See Section 3.11.4.1.2
D	Enable Carry into B1	See Section 3.11.4.1.2
E	Enable Carry into B2	See Section 3.11.4.1.2
F	"0" - Load Counter A Event Selects and Gates (Channel BTA Bits 0-F). "1" - Load Counter B Event Selects and Gates (Channel BTA Bits 0-F). This bit should be set to the proper counter before the count specification is set into Channel BTA7.	

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-15 CHANNEL BTA7

Bit	Function
0	Event Select for Counter A1 and B1 -See Section 3.11.4.1 for codes
1	
2	
3	
4	
5	Event Select for Counter A2 and B2 See Section 3.11.4.1 for codes
6	
7	
8	
9	
A	Not Used
B	Selected Job Gate — MCU Event Counter Gates --See Section 3.11.4.1.1
C	Monitor Mode Gate
D	Job Mode Gate
E	Data Flag 56 Gate
F	Data Flag 57 Gate

(continued)

3.11.1.2 (Cont.)

TABLE 3.11-16 CHANNEL BTA8

Bit	Function
0	8-bit function select code. Bit 0 is the left-most bit of the code. See event code 12_{16} in Section 3.11.4.1.
1	
2	
3	
4	
5	
6	
7	
8	8-bit function mask. Bit 8 is the left-most bit of the mask. See event code 12_{16} in Section 3.11.4.1
9	
A	
B	
C	
D	
E	
F	

3.11.2 MCU/Microcode Memory Interface

Upon power up of the FMP all microcode memory contents are undefined since that memory is built of RAM circuits with volatile storage. Each of the FMP microcodes can be loaded by an MCU function which is sent over the FMP I/O channels from one of the PDCs acting for the MCU. A special trunk address identifies the special I/O channel which does not transfer data to the Intermediate Memory but instead provides control information for the FMP, and retrieves status information from the FMP from one or more of the internal maintenance channels contained within the FMP. One of the maintenance functions is the loading of microcode to each of the microcode memories. Each block of microcode received by the MCU interface is checked for data errors (using the CRC code in the trunk message) and sent to its respective microcode memory system. Each block is preceded by a unique 16-bit address which identifies the particular microcode destination.

3.11.2.1 Microcode Units and Addresses

(to be supplied later)

3.11.2.2 Microcode Error Checking

Under control of the MCU interface control signals, a microcode memory can be loaded with data from the trunk. The data carries its own parity bits (one per word) which are generated by the assembler at the time the microcode is created. This block can be read out of each microcode memory sequentially by the MCU interface so that the memory can be checked. Each word read is parity checked and if an error occurs the location of the failing word is unloaded by the MCU interface via the P counter of that microcode.

During normal startup procedures, each microcode memory is loaded in turn with its unique microcode and the entire contents are swept out on an MCUcontrolled, sequential read operation to verify the integrity of that memory.

(continued)

3.11.2.2 (Cont.)

During operation of the FMP, each microcode access is parity checked. If a parity error occurs in any microcode, the MCU is signalled via the network trunk and the FMP CPU is stopped as soon as possible. The location of the error P counter and the address of the failing microcode unit are then provided to the MCU interface for transmission to the MCU processor on the trunk.

3.11.2.3 MCU Interface Channel Bits

(to be supplied later)

3.11.3 Microcode Memory Channel Programming

The following subsections define microcode interface function codes, switches, sequences, status, etc. as they exist on the STAR-100A. A very similar set will be defined later for the FMP.

3.11.3.1 Typical Microcode Interface Function Codes

For all channel functions the address that accompanies the function and the null function are ignored. The 3-bit function codes shown in Table 3.11-17 control the microcode memory.

TABLE 3.11-17 TYPICAL FUNCTION CODES (MIC. MEM.)

Bit 0	Bit 1	Bit 2	Function
0	0	0	Null - Automatically sent by the MCU interface as the second half of any other function.
0	0	1	Read Memory - Read a block of microcode memory from the current microcode "P" address.
0	1	0	Write Memory - Write a block of microcode memory from the current microcode "P" address.
0	1	1	Not normally used but will perform the same as a EOP.
1	0	0	Data - Automatically sent with the data during a write microcode memory operation.
1	0	1	Read Status - Read the current microcode status. See Section 3.11.3.3 for explanation.
1	1	0	Write Switches - The switches provide control of microcode execution. See Section 3.11.3.2.
1	1	1	EOP - End of Operation clears the interface of all previous functions and also clears the counter that controls the data fan-in and fan-out to/from the channel.

3.11.3.2 Microcode Switches

Microcode switches are 1-bit control terms used to control the microcode memory. Each switch is one bit of the Write Switch Control Word. The 110 function code (write switch) causes the microcode memory to store the Write Switch Control Word in a register. The MCU interface receives this data from the I/O trunk and sends it to the microcode control. The following is a definition of each switch function and a description of its use.

1. Switch Function Definitions

TABLE 3.11-18 MICROCODE SWITCH FUNCTIONS

Bit	Function
0	Go Microcode - Strobing this bit will cause microcode to start execution at the current microcode "P" address.
1	Kill - Setting this bit will stop any microcode instructions executing at the time the bit is set. The instruction will come to a normal halt with "P" pointing to the next word to be executed. Execution can be resumed by setting bit 0.
2	Sense Switch - Any microcode program can sense the condition of this switch for program control (used mainly by diagnostics).
3	P to 0 - Strobing this bit will force the "P" register to zero. Kill should be set either previously or in the same word so as to come to a normal halt.
4	Clear Checkpoint - Strobing this bit will clear the check point flip-flop.
5	Drop Control-Setting this bit disables control of the CPU and the I.C.s from microcode. This will prevent undefined CPU operation due to a microcode memory test.

(continued)

3.11.3.2 (Cont.)

TABLE 3.11-18 MICROCODE SWITCH FUNCTIONS (Cont.)

Bit	Function
6	Change Status Word 2 Definition - Bits 8-F of Status Word 2 become bits 0-7 of an IC register. See Section 3.11.3.3.
7	Enable control of the register logical pipe from microcode.
8	Function for Scalar Microcode not yet defined.
9	Sweep Scalar Microcode
A	Write Scalar Microcode - Must be set to Write. Scalar Microcode, disables microcode write enables.
B	1 = Enables Scalar Microcode to sweep PM00 0 = Enables Scalar Microcode to sweep PM01
C-F	Functions for Scalar Microcode not yet defined.

(continued)

3.11.3.2 (Cont.)

2. Use of Switch Functions

1. Switch Functions 0, 3 and 4 are one-shot functions. This is accomplished by having the required bit set in the even 16-bit word of a transfer and clear in the odd 16-bit word. If the bit is set into both halves of a 32-bit transfer, for instance, the function will be performed in that transfer but will possibly be ignored if sent in the next transfer.

2. Switch Functions 0 and 3 are delayed by one cycle so that other functions sent in the same data word have time to propagate; i.e., "kill" and "P to 0" together are legal as are "sense switch" and "go microcode". Other combinations are also legal.

3. Switch Functions 1, 2, 5, 6, 7, 9, A, and B are latching functions that are caught and held until another function is sent. Note, however, that a single function consists of two or more data transfers -- each transfer clearing and loading over previous data transfers so that a switch that is meant to be valid both during and after the function must be sent in both halves of a 32-bit data transfer and any latching function that is supposed to remain valid through another "send switches" function must be sent again with that function, again present in both halves of the 32-bit data word.

3.11.3.3 Stream Microcode Status

The input of status to the MCU can be of any number of words; but all words after the first word will be word 2 of the status.

The input of status does not have any effect on microcode or microcode controls.

Upon the receipt of a 101 (read status) channel function code, the MCU interface will load the channel with the following status words.

TABLE 3.11-19 MICROCODE STATUS

Bits (Word 1)	Meaning
0	Checkpoint - Software uses this bit to indicate to the MCU that the microcode has reached some predefined status found an error or reach some predefined address for debugging, for example.
1-4	Flags - The current state of flags 0, 1, 2, 3.
5-F	P - The current state of the P (microcode address) register.*
Bits (Word 2)	**Meaning**
0	Run - This bit will be used to indicate the microcode is executing.
1-4	J1 - The current state of the least significant 4 bits of the J1 register.
5-F	J2 - The current state of the J2 register. (See bit 6 of the switch function control word).

*The contents of P do not indicate the address at which microcode has stopped until the second minor cycle after the RUN bit has gone to zero. Thus it is necessary to read the status word twice, once to determine that microcode is not running, and once to read P.

3.11.3.4 Interface Sequences

After selection of the MCU interface the following are examples of possible control sequences.

TABLE 3.11-20 INTERFACE SEQUENCES

Step	Code	Sequence (Stream Units Write Microcode)
A	111	EOP - To clear the interface. Initiate (bit 0) should not be sent with any EOP function.
B	010	Write Mode - The address with this function is ignored; the write will proceed from the current P address.
C	100	Data - Data sent to microcode must (except on the last transfer) be sent in integer multiples of microcode words. One microcode word is 14 16-bit transfers. Data will be lost and/or rearranged if this is not observed.
D	111	EOP
E		Repeat from Step B as many times as necessary to complete transfer of the block of data.
Step	**Code**	**Sequence (Stream Units Read Microcode)**
A	111	EOP
B	001	Read Mode - The address is ignored.
C		Input the data. The same caution as in Write Microcode Step C applies. Data starts from the current microcode P address.
D	111	EOP
E		Repeat from Step B as many times as necessary.

Note: If the last operation performed in a sequence is an EOP, the next sequence does not have to start with another EOP.

(continued)

3.11.3.4 (Cont.)

After selection of the MCU interface the following are examples of possible control sequences.

TABLE 3.11-20 INTERFACE SEQUENCES (Cont.)

Step	Code	Sequence (Stream and Scalar Write Switches)
A	111	EOP
B	110	Set Switch Mode - The address is ignored.
C	100	Data - Although one 32-bit transfer is the normal data length, there is no restriction on data length if the extra data length can be useful - repeated starts for instance.
D	111	EOP
Step	**Code**	**Sequence (Stream Read Status)**
A	111	EOP
B	101	Set Status Mode - The address is ignored.
C		Input Data - All data after the first word is status word 2.
D	111	EOP

Note: If the last operation performed in a sequence is an EOP, the next sequence does not have to start with another EOP.

(continued)

3.11.3.4 (Cont.)

TABLE 3.11-20 INTERFACE SEQUENCES (Cont.)

Step	Code	Sequence (Stream and Scalar Write Switches)
Step	Code	Sequence (Write Scalar Microcode)
A	111	EOP - To clear the interface
B	010	Write Mode - Bits 0-8 of the second 16 bits of the address, selects daughter boards 0-8, respectively. The first 16 bits of the address are ignored. The write will proceed from the current P address.
C	100	Data - Bits 0-3 are Write Enables and bits 4-15 are Data. The microcode address is incremented by one for each 16 bit quantity sent by the MCU.
D		Repeat step C until the selected Auxiliary Board has been loaded (normally 1024 16-bit words).
E	111	EOP
F		Repeat from step B to load other Auxiliary Boards.

Note: If the last operation performed in a sequence is an EOP, the next sequence does not have to start with another EOP.

3.11.3.5 Writing or Sweeping Scalar Microcode Memories

The scalar microcode consists of 5 memories; PM00, PM01, HM00, DM00 and GM00. Although each operates independently during CPU instruction execution, they are all addressed simultaneously during writing or sweeping operations.

3.11.3.5.1 Scalar Microcode Memory Write Operations

For write operations, the write enables at each auxiliary board control which auxiliary board and which address within an auxiliary board is to be written. Since 12 bits of data are written at a time, the write enables are also responsible for choosing which 12-bit portion of a microcode address is to be written.

Under the control of the write enables and auxiliary board select, one auxiliary board is written at a time. The address registers on the auxiliary boards will first be set to 00_{16} and then cycled thru FF_{16} and then back to 00_{16} while writing one-fourth (or twelve bits) of an auxiliary board. The write enable will then change to address the next twelve bits of the particular auxiliary board and the address register will again cycle through all addresses. This operation will occur four times on each of the 9 auxiliary boards.

It is possible, by bringing up all 9 auxiliary board selects and all write enable bits, to write all bits of all auxiliary boards in one write of 100_{16} words (except PM00/PM01). Each 12-bit segment of the 48-bit word will be duplicated. This would be done only as a maintenance aid for pattern generation during either write or sweep operations.

3.11.3.5.2 Scalar Microcode Memory Sweep Operations

Sweeping of the scalar microcode memory is an operation to be done to detect a parity error on any of the 9 microcode auxiliary boards. The operation simply consists of referencing all 9 auxiliary boards simultaneously with the same address register. Since there is one parity bit per auxiliary board per microcode memory, any parity error or errors will be isolated to the failing auxiliary board or boards.

The control signals necessary to perform the sweep operation are Sweep (switch function bit A), Enable PM00/PM01 (switch function bit B) and Clear Fault. Sweep should be enabled during the entire sweep operation. Enable PM00/PM01 selects PM00 or PM01 microcode memories and Clear Fault will clear any parity errors caused by the sweep operation. If Clear Fault is sent while sweep is still set, not only will the parity fault condition be cleared but sweeping will continue. However, since the sweep address, upon a parity fault, is 3, 4, or 5 addresses ahead of the actual parity fault address, sweeping immediately after a parity fault will "skip" 3, 4, or 5 addresses respectively. For example, if the parity fault address is 22 on PM00 then addresses 23, 24, and 25 will be skipped.

The register used to reference all of the auxiliary boards during the sweep operation is cleared before and after the sweep operation. Thus, sweeping starts with address zero and, because of the time delay in detecting parity errors, will end beyond that address which caused the parity error. For example, if the parity error occurred on HM00 at address 125 then the address displayed at the MCU will be 130 and is therefore 5 ahead.

The following list specifies how far ahead of the parity fault address the sweep address will be.

GM00	5 ahead
DM00	4 ahead
HM00	5 ahead
PM00	3 ahead
PM01	3 ahead

3.11.4 Monitoring System Activity by the MCU

The MCU monitors the output of two display registers as its main medium of monitoring system activity. One display register contains the output of the Current Instruction Address Register (CIAR). The other display register contains the output of the register selected by the MCU. A 4-bit code sent from the MCU selects which register the display register will present. In addition to monitoring the display register, the MCU can also monitor the microcode memory status and other CPU status.

3.11.4.1 Monitoring with Counters

For monitoring purposes, the CPU has four 16-bit counters. Each of these counters can be connected to an event line selected by a command from the MCU. See figures 3.11-1 and 3.11-2. A list of events which can be counted and their corresponding select codes is given in Table 3.11-21. For purposes of discussion, one pair of 16-bit counters is referred to as Counters A1 and A2. The other pair is labeled B1 and B2. Counter A and Counter B are completely independent and cannot be tied together; however, they do share the same input event lines and gate lines. The counters can be read by selecting them for input into the MCU display register. They can also be combined in various ways to form one or two 32-bit counters. This reconfiguration is accomplished via the carry lines from the MCU. The counters are enabled by a number of hardware and software gates selected with a mask from the MCU. The MCU has the option of stopping the CPU count condition. This option is exercised by use of the stop lines.

(continued)

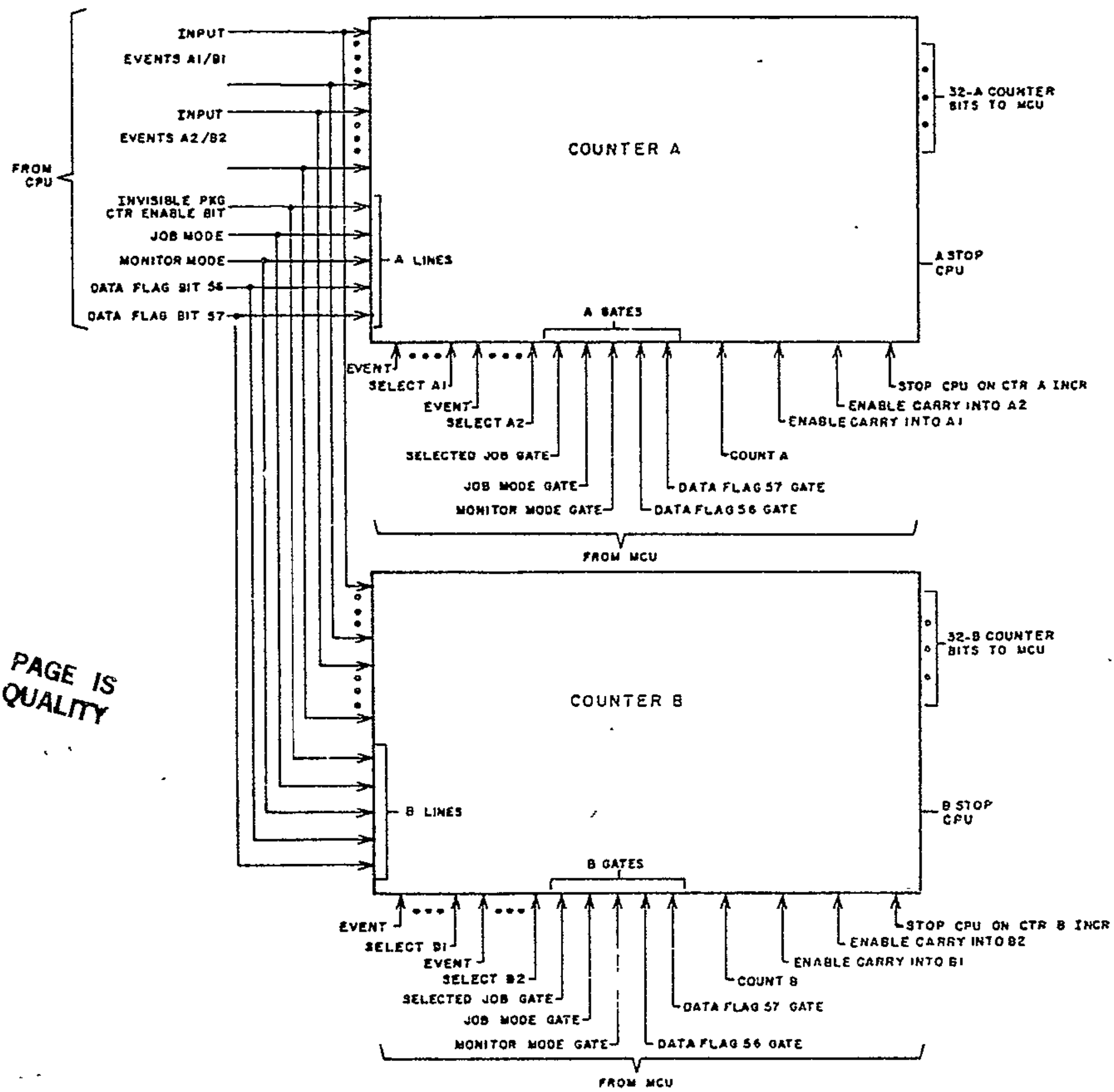


Figure 3.11-1 Block Diagram of Counter Logic Lines

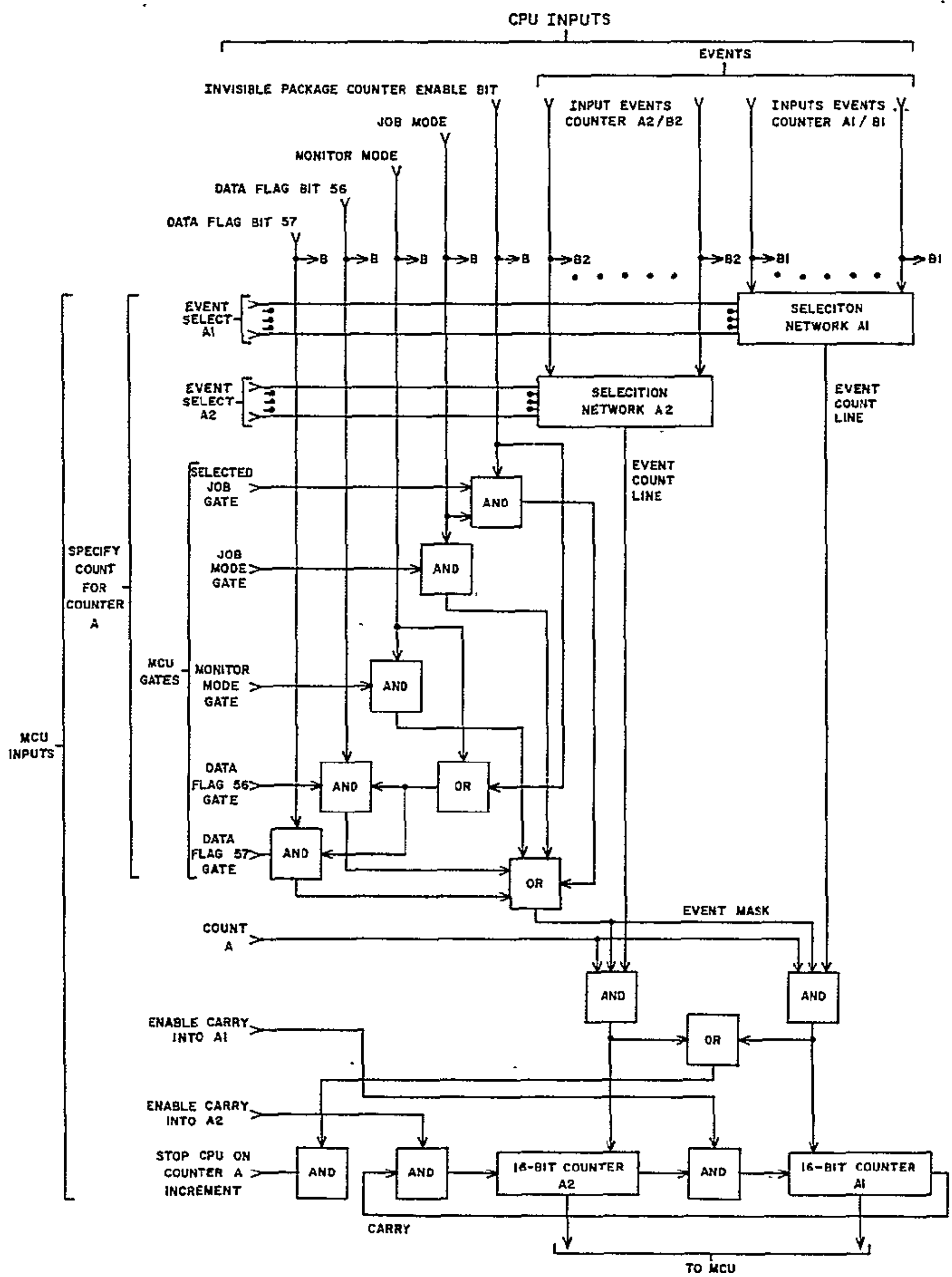


Figure 3.11-2 Block Diagram of Counter A

3.11.4.1 (Cont.)

TABLE 3.11-21 COUNTER EVENTS

*Codes 16		EVENTS
Counter A1/B1	Counter A2/B2	
01		Number of branches out of instruction stack.
	01	Number of branches in instruction stack.
04		Number of times microcode field MON = 1 is selected.
	04	Number of shortstop path usages.
05		Not Used
	05	Not Used.
09		Number of normal channel intermediate memory requests.
	09	Number of normal channel intermediate memory requests accepted.
0A		Number pf CPU memory requests.
	0A	Number of CPU memory requests accepted.
0B		Total number of memory requests.
	0B	Total number of memory requests accepted.
11		Number of minor cycles from selected instruction issue to next, non-selected issue. The counter will begin counting when an instruction whose function code meets the conditions described in code 12 below, is loaded into IRO. It will stop counting when the first following instruction which does NOT meet the conditions is loaded into IRO.

*These are 5-bit codes, expressed in hexadecimal.

(continued)

3.11.4.1 (Cont.)

TABLE 3.11-21 COUNTER EVENTS (Cont.)

*Codes 16 Counter A1/B1	*Codes 16 Counter A2/B2	EVENTS
12		Number of times a particular function code or a particular category of function codes is executed. The count condition is determined by an 8-bit select code and an 8-bit mask sent to the CPU on MCU channel BTA8. If the select code bits and the corresponding instruction function code bits are equal wherever there is "1" in the mask, the counter will be incremented. If the mask contains all zeros, all instructions will be counted.
	12	Time - 1 MHz.
13		Time between selecting microcode monitor field, MON=2 and selecting MON=3.
	13	Number of cycles where data is not available at the output of a functional unit once data has been requested for all input streams. This time does not include the time required for initial setup (preceding the input of the last operands to a functional unit). This count thus permits the programmer to analyze the amount of time required for startup memory accesses, pipeline/functional unit length, and memory conflicts for a specific instruction.

*These are 5-bit codes, expressed in hexadecimal.

Other events will be added, but at the present time codes have not been assigned. Possible additions are:

- Number of scalar requests to Intermediate Memory
- Number of blocks loaded into Backing Store
- Number of blocks read from Backing Store
- Number of map unit references to Intermediate Memory
- Time Swap Unit and Map Units are idle

3.11.4.1.1 MCU Count Gates and CPU Lines

The counters are incremented when the selected event occurs, the count line is up, and one or more of the following gate-line conditions is satisfied:

1. The Event Counter Enable bit is set in the invisible package of the job currently being executed and the Selected Job Gate from the MCU is set. This allows counts to be made during selected jobs only.

2. The CPU is in job mode and the Job Mode Gate from the MCU is set.

3. The CPU is in monitor mode and the Monitor Mode Gate from the MCU is set.

4. Data flag bit 56 (or 57) is set in the Data Flag Register of the CPU and the data flag 56 (or 57) gate from the MCU is set and the CPU is in monitor mode.

5. Data flag bit 56 (or 57) is set in the Data Flag Register of the CPU and the data flag 56 (or 57) gate from the MCU is set and the Event Counter Enable bit is set in the invisible package of the job currently being executed.

There is one set of gate-line enable logic for Counters A1 and A2 and one set for Counters B1 and B2; therefore, Counter A may be enabled by different gates than Counter B.

In summary the CPU lines are:

1. Data flag bit 56.

2. Data flag bit 57.

3. Monitor mode.

4. Job mode.

5. Job enable of monitoring counters from invisible package.

There is a corresponding MCU gate for each of the above.

3.11.4.1.2 Carry Lines

There is one enable carry line associated with each 16-bit counter. Enable carry line A1 enables the carry into Counter A1 from Counter A2. Enable carry line A2 enables the carry into Counter A2 from A1. There are equivalent lines for the B Counter. A zero on carry lines A1 and A2 allows the Counters to operate as two 16-bit counters. Only half of the total number of events are available at the selection network for one Counter A1 or A2; therefore, if a 32-bit count is desired, either counter may have the lower bits of count. For example, if an event is enabled to Counter A1 and a 32-bit count is desired, then enable carry line A1 must equal "0" and enable carry line A2 must be a "1". In this example, Counter A1 will have the least significant bits and Counter A2 will have the most significant.

3.11.4.1.3 Stop Lines

There is one stop line associated with each counter pair, one for the A Counters and one for the B Counters. When the stop line is a "1", an event incrementing either 16-bit counter will stop the computer. Mode line "Event Stop" is returned to the MCU (bit 4, channel ATB8) to show why the CPU has stopped. The MCU, after sending a "Clear Fault Signal", may restart the CPU.

3.11.4.1.4 Counter Setup

Typically, the four counters would be set up by the MCU as follows:

1. Set the following bits as required:

    a. Stop CPU on A Increment (bit 9, channel BTA6)
    b. Stop CPU on B Increment (bit A, channel BTA6)
    c. Enable carry into A1 (bit B, channel BTA6)
    d. Enable carry into A2 (bit C, channel BTA6)
    e. Enable carry into B1 (bit D, channel BTA6)
    f. Enable carry into B2 (bit E, channel BTA6)

2. With bit F, channel BTA6, a zero, set event and mask selection for Counter A into channel BTA7.

3. Set bit F, channel BTA6 to a "1".

4. Set event and mask selection for Counter B into channel BTA7.

(continued)

3.11.4.1.4 (Cont...)

5. If A1/B1 event code 12 for function counting has been selected, set channel BTA8 to the desired function and mask.

6. Set count line A or B (bit 6 or 7, channel BTA6) as desired.

The counters will now be counting events and will continue to count until their respective count lines are dropped.

3.11.4.2 Display Registers

There are two 64-bit display registers that can be monitored by the MCU. One display register is used for the Current Instruction Address Register (CIAR) and the other is used for a register that has been selected by the MCU. The register is selected by a 4-bit code transmitted on bits C-F of channel BTA1. Any unlisted bits (such as bits 0-16 for code 3) are undefined.

The MCU must send a read signal to enable the CIAR and the selected register into the display registers. The read signal has been defined as bit B on channel BTA1 and its leading edge simultaneously transfers both registers into the display registers. The register select code must be set up by the MCU before the read signal is transmitted to the CPU.

The CIAR is received on channels ATB1 - ATB3 of the MCU and may read while the CPU is running. The selected register is received on channels ATB4 - ATB7 of the MCU. See Section 3.11.1.1 for bit assignments. The selected register on channels ATB4 - ATB7 may only be read when the CPU is stopped.

The select codes and corresponding registers are listed in the following table (many registers have not had codes assigned and are therefore not yet shown; as design proceeds the table will be expanded.

(Continued)

3.11.4.2 (Continued)

TABLE 3.11-22 DISPLAY REGISTER SELECT CODES

Code 16	Register(s)	Bits
0	Current Instruction Register	0-63
1	Data Flag Register	3-15
		19-31
		35-47
		31-58
2	Invisible Package Address	0-22
	(Absolute Sword Address)	
3	External Interrupt Register	15-30
	Monitor Interval Timer	15
	Channel 0	16
	1	17
	1	17
	2	18
	3	19
	4	20
	5	21
	6	22
	7	23
	8	24
	9	25
	10	26
	11	27
	12	28
	13	29
	Swap Unit	30
	Channel Read Active - Write Active	32-61
	Channel 0	32-33
	1	34-35
	2	36-37
	3	38-39
	4	40-41
	5	42-43
	6	44-45
	7	46-47
	8	48-49
	9	50-51
	10	52-53
	11	54-55
	12	56-57
	13	58-59
	Swap Unit	60-61

(continued)

3.11.4.2 (Cont.)

TABLE 3.11-22 DISPLAY REGISTER SELECT CODES (Cont.)

Code 16	Register(s)	Bits
4	SECDED Fault Read Bus Code	0-3
	Instruction Stack Parity Fault	4
	MIC Memory 0 Parity Fault	5
	MIC Memory 1 Parity Fault	6
	Scalar MIC Parity Fault	7
	Double Secded Error. Syndrome Bits must be checked to determine if address and Bus Code are valid.	8
	Syndrome Bits	9-15
	Parity Fault on Auxiliary Board 0	16
	Parity Fault on Auxiliary Board 1	17
	Parity Fault on Auxiliary Board 2	18
	Parity Fault on Auxiliary Board 3	19
	Parity Fault on Auxiliary Board 4	20
	Parity Fault on Auxiliary Board 5	21
	Parity Fault on Auxiliary Board 6	22
	Parity Fault on Auxiliary Board 7	23
	Parity Fault on Auxiliary Board 8	24
	PM01 Enabled for Parity Checking	25
	Scalar Microcode Address -Bit 0	26
	Scalar Microcode Address -Bit 1	27
	Scalar Microcode Address -Bit 2	28
	Scalar Microcode Address -Bit 3	29
	Scalar Microcode Address -Bit 4	30
	Scalar Microcode Address -Bit 5	31
	Scalar Microcode Address -Bit 6	32
	Scalar Microcode Address -Bit 7	33
	NOTE: All Fault/Error conditions are cleared by the "Clear Fault" signal from the MCU except the SECDED Error and the Syndrome bits. These are cleared/released by the "Clear Single Error" signal from the MCU.	

(continued)

3.11.4.2 (Cont.)

TABLE 3.11-22 DISPLAY REGISTER SELECT CODES (Cont.)

Code 16	Register(s)	Bits
4	(continued)	
	SECDED Fault Address (Absolute physical bit address, significant to the half-word level)	34-63
	The address of the first SECDED error is retained in this register.	
	The SECDED Fault Address is released by the Clear Single Error Condition Signal from the MCU.	
5	Bounds Hit Address (Absolute physical bit address, right justified) The address of the first bounds hit is retained in this register. The bounds hit address is released by the Clear Fault Condition Signal from the MCU. The bounds checking is performed on half-word boundaries only.	0-31
6	Counter A1	0-15
	Counter A2	16-31
	Counter B1	32-47
	Counter B2	48-63
	If bit 8 of channel BTA6 in the MCU is a "0", both counters will be cleared after the read signal is received and after both counters are transferred into the display register. If bit 8 is a "1", the counters will not be cleared.	
	To ensure proper initialization of the counters, the count lines must be made zero prior to the new count selection.	

3.11.4.3 Logic Fault Monitoring

There are several types of logic faults detected in the computer such as memory SECDED, MIC memory parity, pipeline compare, and I/O parity. When a logic fault is detected, the computer stops between instructions. The types of fault may be sensed on channel ATB8. (See Section 3.11.1).

After sensing the logic fault, the MCU must clear the fault via bit 7 of channel BTA1. The MCU must determine the appropriate response to the fault and has the option of restarting the CPU by setting bit 3 of channel BTA1.

3.11.5 SECDED (Single Error Correction Double Error Detection)

Single error correction/double error detection (SECDED) is carried on all data buses within the FMP. SECDED is generated on all information as it enters the FMP and is regenerated whenever the form of the data is modified (as an arithmetic result). SECDED is checked and any errors corrected before the data is put into or used by a possibly failure-prone element (such as memory) or an element that will modify the data (such as the Vector Unit). SECDED check networks are placed at intervals in the data path which permit immediate identification of a failing memory or functional element in most cases.

SECDED is carried in two forms in the FMP. The first form is 7 SECDED bits with each 32 data bits. Data in this form is kept and supplied by all the units associated with the Main Memory, including Main Memory itself. This is because the least addressable element in Main Memory is a half-word of 32 bits and the bandwidth requirements prohibit frequent changes of SECDED. See Table 1 of appendix A for 32-bit SECDED syndromes.

The second form is 8 SECDED bits with 64 data bits. Data in this form is used by the Backing Store, the Intermediate Memory and their associated units--the Intermediate Map Unit and the Swap Unit. See table 2 of appendix A for 64-bit SECDED syndromes. Where there is an interface between the two different SECDED forms, SECDED check/regenerate logic translates the SECDED syndromes after checking and possibly fixing the associated data.

(continued)

3.11.5 (Cont.)

The processor keeps information on SECDED errors as they occur in the machine. Separate registers record errors found by SECDED checking logic for the two different SECDED forms. The data kept includes the faulty syndrome as well as the location of its occurrence and type of error, i.e., a single (corrected) bit error or a multiple (uncorrected) bit error. When an error occurs, the Maintenance Control Unit (MCU) records the error on its local disk (the system error log) and clears the error registers. If other, similar SECDED errors occur that would record their error information in the same set of registers they are lost. (However, since the processing system is expected to have a very low failure rate, loss of errors should be negligible.)

When the processor finds a single-bit error the SECDED logic corrects the failing bit and the processor continues as before with no loss in throughput. (The error is, of course, recorded.) When a double-bit (or worse) error occurs, what happens depends on whether or not the machine was in job (user) mode or not. If the FMP was in monitor mode the error is recorded by the MCU which will then restart the FMP. Any jobs within the FMP at the time will or will not be lost depending upon the ability of the operating system to do a "warm" start. If the FMP was in job mode, the processor forces the machine back into the monitor (monitor mode) which will abort the job but continue to run.

There is one place in the FMP that does not carry SECDED--the I/O Unit. Each PDC carries data parity internally and also on the connections between the PDCs and Intermediate Memory. This causes very little loss because the PDC is an intelligent device that can request or supply data again when a data error is detected.

The SECDED error information is logged by the Maintenance Control Unit (MCU). The information logged is syndrome word, single error, double error, Read bus code, and CPU word address bits 37-58.

(continued)

3.11.5 (Cont.)

SECDED ERROR INFORMATION

1. SYNDROME BITS - These seven bits generated by the error correcting code . The 39 unique syndrome words for single bit errors are listed in appendix A. Of these 39 (odd bit) syndrome words, only the 32 data bit codes will toggle a bit when error correction is enabled. Other odd bit codes latched in SECDED that differ from the 39 unique syndrome words will be flagged by the MCU as a multiple odd bit error. Double error syndrome words have an even number of bits.

2. SINGLE ERROR - Bit 5 of channel ATB8 (see section 3.11.1) will set if there is a single error <u>not</u> <u>preceded</u> by a double error.

3. DOUBLE ERROR - This MCU display register will set unconditionally on a double error.

4. SECDED FAULT BUS CODE - These MCU display registers define the read bus on which the SECDED error occurred.

To Be Defined Later

5. HALF-WORD ADDRESS (Bits 57,58) - These address bits decode the four 32-bit groups within a quarter sword.

6. CPU WORD ADDRESS (Bits 37-56) - These address bits indicate the following:

To Be Supplied Later

7. LATCHED ADDRESS BITS (37-58) - In SECDED these address bits are always the physical CPU Word Address Bits.

3.11.6 Absolute Bounds Address

The absolute bounds address mechanism provides the facility to notify the MCU of a memory reference (read or write) inside a specified block of memory. The block of memory is specified by an upper bounds address and a lower bounds address. Note that the addresses are absolute physical addresses transmitted from the MCU. The bounds addresses are defined as not included in the block of memory.

The checker can selectively test various classes of requests for in-bounds conditions. Any combination of classes may be selected.

If the FMP has been stopped by a bounds hit, the hit must be cleared by the clear fault signal from the MCU before the FMP can be restarted. The FMP can be restarted to execute the next instruction in sequence.

The occurrence of a bounds hit (i.e., a selected memory reference inside bounds) is signaled to the MCU. To identify a second bounds hit, the MCU must clear the first bounds hit signal via the clear fault signal.

When a bounds hit is made, the address of the causing request is saved in the bounds hit register until a Master Clear or Fault Clear occurs.

The bounds limits and the bounds hit address refer to physical addresses, which are independent of all Memory Degradation modes. (The bounds test is applied to the address after any Degradation mode manipulation has been applied).

3.12 Timing Information

The FMP is in preliminary design phase so only the most preliminary timing estimates are available. All estimates are given in CPU minor clock cycles. The period of this clock cycle is expected to be 16 nanoseconds.

3.12.1 Scalar Unit Timing

The table in Section 3.12.1.2 is designed to provide scalar timing data for instruction sequences in the FMP. All timing data is expressed in minor cycles.

Multi-operand instructions are typically expressed as overhead + (number of cycles per operand) (number of operands).

3.12.1.1 Use of Scalar Timing Tables

The ISSUE portion of the table gives the minimum number of minor cycles between the issue of the specific instruction listed in the left column and the issue of the next instruction in a program sequence. Various operand or memory conflicts (as discussed later) can cause additional delay beyond This minimum time.

The Issue portion of the tables is sub-divided when appropriate into three categories as defined below:

NB -- No Branch
ISB -- In Stack Branch
OSB -- Out of Stack Branch to first quartersword. This time must be increased by 1, 2, or 3 if the Branch address is in the 2nd, 3rd, or 4th quarter-sword, respectively.

The non-branch instructions use the entry under NB or No Branch. Example 1 illustrates a simple, no conflict, Branch sequence.

Example 1

	Instr.	R S T	Comments
A)	60	- - -	Register designators which are not pertinent to the example are represented by a -.
B)	25	- - -	Branch condition not met.

	Instr.	R S T	Comments
C)	65	- - -	
D)	25	- - -	Branch condition met, branch out of stack to instruction E in 3rd quarter-sword.

Sequence Timing

Instr. A issues at minor cycle 0
Instr. B issues at minor cycle 1
Instr. C issues at minor cycle 12
Instr. D issues at minor cycle 13
Instr. E issues at minor cycle 42

(continued)

3.12.1.1 (Cont.)

The RESULT AVAILABLE portion of the table contains information necessary to time instruction sequences with operand dependencies. The first column, SS or shortstop, contains entries for those instructions which use the Scalar Floating Point. These are the instructions which may use the shortstop feature to provide an input operand. This entry is the number of minor cycles after issue that the result operand will be available at the shortstop for use with a following instruction.

If instruction A issues at minor cycle X, any following instruction, B, needing the result of A must issue no later than minor cycle X+SS to utilize the shortstop. A floating-point instruction needing the result of A, can be issued before X+SS and wait at the input of Floating Point for the shortstopped result of A. This allows other non-floating-point instructions to issue. The resulting time of an instruction that issues and waits for shortstop will be as if it had issued at the ideal time to match shortstop. A subsequent instruction requiring access to Floating Point will not issue any earlier than {X+SS} + 1.

If instruction B issues later than cycle X+SS, thus missing the shortstop, instruction B must wait until at least X+RF. At this time the desired operand will be available from the Register File. Example 2 illustrates operations using shortstop.

Example 2

	Instr.	R	S	T	Comments
A)	60	-	-	12	
B)	60	-	-	-	
C)	60	-	-	-	
D)	60	-	-	-	
E)	60	-	-	-	
F)	60	12	-	14	
G)	60	14	14	-	
H)	7F	-	-	-	
I)	60	-	-	15	
J)	60	14	-	16	
K)	60	16	15	-	
L)	60	-	-	-	

(continued)

3.12.1.1 (Cont.)

Sequence Timing

```
Instr. A issues at 0
Instr. B issues at 1
Instr. C issues at 2
Instr. D issues at 3
Instr. E issues at 4
Instr. F issues at 5  -- Thus exactly matching shortstop

Instr. G issues at 6  -- Issues but must wait at the input
                         of Floating Point for the
                         result of instruction
                         F to be available
Instr. H issues at 7

Instr. I issues at 11 -- Cannot issue until instruction G
catches shortstop and proceeds.
                             Thus 3 minor cycles not used

Instr. J issues at 13 -- Missed result of instruction F at
                         shortstop thus waiting until
                         operand is available from
                         Register File

Instr. K issues at 14 -- Issues and waits at input to
                         Floating Point

Instr. L issues at 19 -- Instruction K is treated as if
                         issued at 18 and the L at 19
```

The last column under RESULT AVAILABLE (MEM) contains entries for those scalar instructions (13, 32, 5F, 7F) which store a result into Main Memory. The time listed is the minimum time until the operand is in memory and available for use. The time may also be increased by 4 minor cycles if the desired memory bank is busy.

(continued)

3.12.1.1 (Cont.)

The UNIT BUSY portion of the table concerns instructions issued to either Divide/Convert (D/C) or Main Load/Store (L/S). Divide/Convert executes the 10, 11, 4C, 4F, 53, 6C, 6F and 73 instructions. This unit is the only portion of Scalar Floating Point which is not completely pipelined; thus the appropriate unit busy time listed in the table must elapse before a third instruction can be issued to Divide/Convert. Floating point instructions other than these eight may be issued to Floating Point while Divide/Convert is busy. A second instruction from the set of eight can be issued, but will be held in front of Scalar Floating Point and issuing of non-floating-point instructions will continue.

Main Load/Store executes the 12, 13, 32, 5E, 5F, 7E, and 7F instructions. There are six address registers in Load/Store which enable requests to be stacked and executed in the proper order. The 12, 5E, and 7E instructions each require one register and can be executed (in the absence of memory conflicts) at the rate of one load per minor cycle. The 5F and 7F instructions each require two address registers and can be executed at one store per two minor cycles. The 13 and 32 instructions each require two address registers and can be executed at one per 14 and 15 minor cycles, respectively.

Main Load/Store is then capable of streaming Load/Store instructions (other than the 13 and 32) at one minor cycle per load and two minor cycles per store assuming no Memory or Register File conflicts. For example, a stream of N loads will execute in N + 14 minor cycles from the issue of the first load until the operand from the last load is available in the Register File. A stream of N stores will execute n $2N + \left\lceil \frac{N}{3} \right\rceil$ minor cycles from issue of the first store until issue of the last store.

Example 3

	Instr.	R	S	T	Comments
A)	60	-	-	-	
B)	7E	-	-	-	
C)	13	-	-	-	
D)	13	-	-	-	
E)	60	-	-	-	
F)	7E	-	-	-	
G)	7E	-	-	-	
H)	7E	-	-	-	
I)	13	-	-	-	

(continued)

3.12.1.1 (Cont.)

Sequence Timing

Instr. A issues at 0	
Instr. B issues at 1	
Instr. C issues at 2	
Instr. D issues at 4	
Instr. E issues at 6	
Instr. F issues at 7	
Instr. G issues at 8	
Instr. H issues at 19	Instr. H must wait for address register to become free from Instr. C.
Instr. I issues at 35	Instr. I must wait for address register.

There are three additional Operand Dependencies which must be considered.

1. Source operand conflict -- an instruction requiring the result of a previous instruction as an input operand waits until the operand becomes available.

2. Output operand conflict -- an instruction output to the same Register File location as a previously issued, but slower instruction, waits until the previous instruction stores its result in the Register File.

3. Register File Write conflict -- an instruction cannot issue if its result arrives at the Register File at the same minor cycle as the result of a previously issued but slower instruction.

Table 3.12-1 pertains to instructions having greater than 1 minor cycle issue time.

The first column lists the appropriate instructions. The second column indicates the minor cycle of issue that a specific operand is required. The third column indicates the availability of shortstop for that specific operand.

(continued)

3.1.2.1.1 (Cont.)

TABLE 3.12-1. ORDER THAT DESIGNATORS ARE READ FOR MULTIPLE ISSUE INSTRUCTION AND IF THEY CAN CATCH SHORTSTOP OF A PREVIOUS INSTRUCTION

INSTRUCTION		DESCRIPTION	SHORTSTOP
13	1st	R&S	No
	2nd	T	No
21	1st	R&R+1	Yes
	2nd	S	Yes
25&27	1st	R&S	No
	2nd	T	No
2F	1st	S	No
	2nd	T	No
	3rd	T	No
31&35	1st	S&T	No
	2nd	R	No
	3rd	R	No
32	1st	S	No
	2nd	T	No
36	1st	S&T	No
	2nd	R	No
	3rd	R	No
5F	1st	R&S	No
	2nd	T	No
6D	1st	R&S	Yes (R+S)
	2nd	T	No
7F	1st	R&S	No
	2nd	T	No
B0-B5.X00X-X	1st	B&Y	No
	2nd	X&A	No
	3rd	Z	No
	4th	X&A&C	No
B0-B5.X01X-X	1st	B&Y	No
	2nd	X&A	Yes (X+A)
	3rd	Z	Yes
B0-B5.X10X-X	1st	B&Y	No
	2nd	X&A	Yes
B0-B5.X11X-X	1st	B&Y	No
	2nd	X&A	Yes
B6	1st	R	No

(continued)

3.12.1.1 (Cont.)

Example 4

	Instr.	R	S	T	Comments
A)	60	-	-	12	
B)	36	10	-	12	Specifies an out-of-stack branch to Instruction C in the 2nd quarter-sword
C)	60	-	-	35	
D)	B0	40	35	-	Specifies an in-stack branch to Instruction E
E)	60	-	-	-	

Sequence Timing

Instr. A issues at 0	
Instr. B issues at 8	B must wait for Result from A to be stored into the Register File
Instr. C issues at 32	
Instr. D issues at 36	Result from Instruction C available from Shortstop at time 37 allows issue at 36
Instr. E issues at 48	

3.12.1.2 Basic Instruction Timing

TABLE 3.12-2 SCALAR INSTRUCTION TIMES

	Issue			Result Avail.			Unit Busy	
Instructions	NB	ISB	OSB	S.S.	R.F.	MEM	L/S	D/C
00	Waits for external or real-time interrupt							
04	20							
06	4							
08	20							
09	Job to Monitor							
	349	--			--	--		
	Monitor to Job							
	324	--			--	--		
0A	20							
0E		20	34					
10	1	--	--	22	25			18
11	1	--	--	53	56			49
12	1	--	--	--	15		1*	
13	2	--	--	--	--	23	14*	
20	1	--	--	4	7	--		

* MUST ADD 5 MC FOR REGISTER RELEASE

(continued)

3.12.1.2 (Cont.)

TABLE 3.12-2 SCALAR INSTRUCTION TIMES (Cont.)

	Issue			Result Avail.			Unit Busy	
Instructions	NB	ISB	OSB	S.S.	R.F.	MEM	L/S	D/C
21	2	--	--	5	8	--		
24	1	--	--	--	*	--	*	
25	2	--	--	--	--	*	*	
26	1	--	--	--	*	--	*	
27	2	--	--	--	--	*	*	
2B	1	--	--	3	6	--		
2C	1	--	--	3	6	--		
2D	1	--	--	3	6	--		
2E	1	--	--	3	6	--		
2F	7	8	23	--	7	--		
30	1	--	--	3	6	--		
31	7	8	23	--	7	--		

* Time is yet to be established.

(continued)

3.12.1.2 (Cont.)

TABLE 3.12-2 SCALAR INSTRUCTION TIMES (Cont.)

	Issue			Result Avail.			Unit Busy	
Instructions	NB	ISB	OSB	S.S.	R.F.	MEM	L/S	D/C
							G-bits	2&3
32.00X-X	2	--	--	--	--	(24	15*)	0 1
32.01X-X		9	24	--	--	(24	15*)	1 0
32.1X-X	20	21	36	--	--	(24	15*)	1 1
33.XXXXX0XX								
33.XXXXX1XX								
34	1	--	--	3	6	--		
35	7	8	23	--	7	--		
36,R=T,S=0	5	--	--	--	5	--		
36,R=T,S≠0		9	24	--				
36,R≠T		8	23	--	5	--		
37	32			--	--			
38	1	--	--	1	4	--		
39	30			--	--			
3A	20			--	--			
3B	26	--	--		--	--		
3C	1	--	--	5	8	--		
3D	1	--	--	5	8	--		

* MUST ADD 5 MC FOR REGISTER RELEASE

(continued)

3.12.1.2 (Cont.)

TABLE 3.12-2 SCALAR INSTRUCTION TIMES (Cont.)

	Issue			Result Avail.			Unit Busy	
Instructions	NB	ISB	OSB	S.S.	R.F.	MEM	L/S	D/C
3E	1	--	--	1	4	--		
3F	1	--	--	1	4	--		
40	1	--	--	5	8	--		
41	1	--	--	5	8	--		
42	1	--	--	5	8	--		
44	1	--	--	5	8	--		
45	1	--	--	5	8	--		
46	1	--	--	5	8	--		
48	1	--	--	5	8	--		
49	1	--	--	5	8	--		
4B	1	--	--	5	8	--		
4C	1	--	--	30	33	--		26
4D	1	--	--	1	4	--		
4E	1	--	--	1	4	--		
4F	1	--	--	30	33	--		26
50	1	--	--	5	8	--		
51	1	--	--	5	8	--		
52	1	--	--	5	8	--		
53	1	--	--	29	32	--		26

(continued)

3.12.1.2 (Cont.)

TABLE 3.12-2 SCALAR INSTRUCTION TIMES (Cont.)

	Issue			Result Avail.			Unit Busy	
Instructions	NB	ISB	OSB	S.S.	R.F.	MEM	L/S	D/C
54	1	--	--	5	8	--		
55	1	--	--	5	8	--		
58	1	--	--	1	4	--		
59	1	--	--	5	8	--		
5A	1	--	--	3	6	--		
5B	1	--	--	3	6	--		
5C	1	--	--	5	8	--		
5D	1	--	--	5	8	--		
5E	1	--	--	--	15	--	1*	
5F	2	--	--	--	--	10	2*	
60	1	--	--	5	8	--		
61	1	--	--	5	8	--		
62	1	--	--	5	8	--		
63	1	--	--	1	4	--		
64	1	--	--	5	8	--		
65	1	--	--	5	8	--		
66	1	--	--	5	8	--		
67	1	--	--	1	4	--		

* MUST ADD 5 MC FOR REGISTER RELEASE

(continued)

3.12.1.2 (Cont.)

TABLE 3.12-2 SCALAR INSTRUCTION TIMES (Cont.)

	Issue			Result Avail.			Unit Busy	
Instructions	NB	ISB	OSB	S.S.	R.F.	MEM	L/S	D/C
68	1	--	--	5	8	--		
69	1	--	--	5	8	--		
6B	1	--	--	5	8	--		
6C	1	--	--	54	57	--		50
6D	2	--	--	4	7	--		
6E	1	--	--	3	6	--		
6F	1	--	--	54	57	--		50
70	1	--	--	5	8	--		
71	1	--	--	5	8	--		
72	1	--	--	5	8	--		
73	1	--	--	53	56	--		49
74	1	--	--	5	8	--		
75	1	--	--	5	8	--		
76	1	--	--	5	8	--		
77	1	--	--	5	8	--		
78	1	--	--	1	4	--		
79	1	--	--	5	8	--		
7A	1	--	--	3	6	--		

(continued)

3.12.1.2 (Cont.)

TABLE 3.12-2 SCALAR INSTRUCTION TIMES (Cont.)

	Issue			Result Avail.			Unit Busy	
Instructions	NB	ISB	OSB	S.S.	R.F.	MEM	L/S	D/C
7B	1	--	--	3	6	--		
7C	1	--	--	3	6	--		
7E	1	--	--	--	15	--	1*	
7F	2	--	--	--	---	10	2*	
B0.X00 X-X	8	9	24	--	8	--		
B0.X01 X-X	3	--	--	5	5+8**	--		
B0.X10 X-X	11	12	27	--	--	--		
B0.X11 X-X	2	--	--	6	9	--		
B1.X00 X-X	8	9	24	--	8	--		
B1.X01 X-X	3	--	--	5	5+8**	--		
B1.X00 X-X	11	12	27	--	--	--		
B1.X11 X-X	2	--	--	6	9	--		
B2.X00 X-X	8	9	24	--	8	--		
B2.X01 X-X	3	--	--	5	5+8**	--		
B2.X10 X-X	11	12	27	--	--	--		
B2.X11 X-X	2	--	--	6	9	--		

* MUST ADD 5 MC FOR REGISTER RELEASE.

**Output to be stored in Register C is available at 5 cycles and Y at 8 cycles. Y may be used from the Shortstop at time 5. C can not be shortstopped.

(continued)

3.12.1.2 (Cont.)

TABLE 3.12-2 SCALAR INSTRUCTION TIMES (Cont.)

	Issue			Result Avail.			Unit Busy	
Instructions	NB	ISB	OSB	S.S.	R.F.	MEM	L/S	D/C
B3.X00 X-X	8	9	24	--	8	--		
B3.X01 X-X	3	--	--	5	5+8**	--		
B3.X10 X-X	11	12	27	--	--	--		
B3.X11 X-X	2	--	--	6	9	--		
B4.X00 X-X	8	9	24	--	8	--		
B4.X01 X-X	3	--	--	5	5+8**	--		
B4.X10 X-X	11	12	27	--	--	--		
B4.X11 X-X	2	--	--	6	9	--		
B5.X00 X-X	8	9	24	--	8	--		
B5.X01 X-X	3	--	--	5	5+8**	--		
B5.X10 X-X	11	12	27	--	--	--		
B5.X11 X-X	2	--	--	6	9	--		
B6	7	8	23	--	--			
BE	1	--	--	1	4			
BF	1	--	--	1	4			
CD	1	--	--	1	4			
CE	1	--	--	1	4			

**Output to be stored in Register C is available at 5 cycles and Y at 8 cycles. Y may be used from the Shortstop at time 5. C can not be shortstopped.

(continued)

3.12.1.2 (Cont.)

TABLE 3.12-2 SCALAR INSTRUCTION TIMES (Cont.)

Instruction	Execution Time
7D	130 + 12 x (ceiling((N+K)/12)) where: N = Number of elements in the longer of R or T R = Number of elements in the input (R) field T = Number of elements in the output (T) field K = 6 if R > T, 8 if R = T, or 16 if R < T

3.12.2 Vector Processor Timing

All vector processing times are stated in terms of overhead (the time required to start up a vector operation) and vector throughput in results per clock cycle. Total time for a vector operation is then stated as O+N/R where O is the overhead, R is the rate of results per minor cycle, and N=M times the ceiling of L/M; L is the vector length and M is 8 for 64-bit operands or 16 for 32-bit operands. Ceiling is the APL operator which returns the maximum integer value of the argument.

Vector overhead is variable depending on a number of conditions. Its component parts are:

1. Issue time---Instruction Issue requires a certain number of cycles to translate and issue the vector instructions. Issue time includes the time to access required data from the Register File.

2. Transmission of data and control information from the Scalar Unit to the Streaming Control Unit.

3. Checking the dependency flags and waiting for any flag conflicts to clear.

(continued)

3.12.2 (Cont.)

4. Transit time through the vector streaming queue.

5. Transmission of setup data to the Vector Streaming Unit.

6. Vector Streaming Unit setup----The time required to form addresses and initiate the memory requests within the Vector Streaming Unit.

7. Transmission of memory request.

8. Memory access time.

9. Data transmission to Vector Streaming Unit.

10. Data transmission through Vector Streaming Unit.

11. Data transmission to Vector Unit.

12. Time through vector pipelines.

13. Transmission to Vector Streaming Unit.

14. Data path through Vector Streaming Unit.

15. Transmission to memory.

When two vector operations appear in sequential instructions in Instruction Issue, the time in the Streaming Control Unit, time in the Vector Streaming Unit, and various transmission times can be hidden, or overlapped, thus reducing the apparent overhead time. It is possible, in very many cases, for the vector overhead time to be reduced into the range of four to six clock cycles.

As design of the FMP proceeds, the timing of instructions becomes ever more complex. Instruction timing <u>cannot</u> be presented in a simple table with a set of rules for its use. The machine is simply too complicated. What will be done instead is to show best case times or equations for the different instructions or classes of instructions. Also listed will be things that will prevent an instruction from obtaining its maximum rate or minimum execution time.

Before the timing of instructions is attempted a short discussion of vector set time will be given.

(continued)

3.12.2 (Cont.)

ALL machines have setup time. For example, on a scalar machine running FORTRAN, the following code to perform an addition is typical:

1) Fetch A
2) Fetch B
3) Add C=A+B
4) Normalize C (not required for some machines)
5) Store C

Here 4 of the 5 instructions are overhead (setup) time.

On a vector operation the setup time is taken to be: memory access time and depth (in terms of time) of the pipelines. This is true of all vector machines; however, with proper hardware design it might be possible to start another vector immediately behind another. The effect of this is to reduce the effective startup time -- immediately after the last operand of the first vector has been stored to memory, the first operand of the second vector is being stored. The FMP will be able to perform in this mode of operation, and will, in some vector operations, perform with zero effective startup time.

It is, of course, possible that some code sequences cannot use this design feature. If a map operation is issued, counting on data from an immediately previous vector instruction, the full pipeline depth will be seen by the waiting map instruction. The problem can be at least partially relieved by coding so that the immediate dependency is removed by placing other instructions between the vector and map instructions.

Appendix D provides a more detailed explanation of the complexities involved in determination of execution times.

(continued)

3.12.2 (Cont.)

Table 3.12-3 gives Vector Processor startup times.

TABLE 3.12-3 VECTOR STARTUP TIMES

Component Function	Nominal/ Minimum Number of Clock Cycles	Notes
Issue time for 9E/9F instruction	4	(1)(2)
Time in Streaming Control Unit	3	(1)(3)
Time setting up read ports in Vector Streaming Unit	3	(1)
Memory interchange time	3	(1)(4)
Memory access time	3	(1)
Data transmission time	3	(1)(5)
Time through Vector Streaming Unit to Vector Ensemble	3	(1)(6)
Time waiting for previous vector to unwind	0	(1)(7)
Time through Vector Ensemble	9	(1)(8)
Time through Vector Streaming Unit on way to memory	3	(1)
Memory interchange time	3	(1)(4)

Notes:

(1) Time may be hidden under previous vector instructions.

(2) Time may be extended waiting for:

a) results to be available in the register file from load instructions and arithmetic computations;

b) a previous streaming instruction to enter the streaming instruction queues because of a flag conflict.

While b) holds up execution of the vector instruction, the time spent waiting is not counted against the startup time for the current instruction but rather against the instruction that caused the wait.

(3) Time may be extended waiting for:

a) read or write flags to clear that are in conflict with the keys of the current instruction;

b) instructions in the vector streaming queue to be issued to the Vector Processor.

While b) holds up the vector instruction from issuing, it is not counted in the startup time.

(4) Time may be extended by the priority element holding the request or by the requested bank being busy. Normally only the extended time will be seen as startup time (time over 3 cycles).

(5) Transmission time includes time for:

a) address request to Memory Interchange,

b) data from Memory Interchange to read port,

c) data from write port to Memory Interchange.

(6) Data may reside in the read FIFO buffers for some time but this is not counted against startup time.

(7) This is time spent waiting for data paths in the Vector Ensemble to clear - 0, 3, or 6 clock cycles. This time is not counted against startup time.

(8) Time from the first element entering a vector pipeline until the first element exits the pipeline. This time varies depending on the operation sequence being performed.

3.12.3 Map Unit Timing

The Map Unit functions of MERGE, MASK, COMPRESS, SCATTER, and GATHER operations are incorporated physically within the Map Units. Table 3.12-4 gives timing information for these Map Unit functions.

TABLE 3.12-4 MAP UNIT TIMES

Component Function	Nominal/ Minimum Times (cycles) Main Map	Inter. Map	Notes
Issue time for 9D instruction	2	2	(1) (2)
Time in Streaming Control Unit	3	3	(1) (3)
Time setting up read ports in Map Unit	3	3	(1)
Memory interchange time	3	2	(1) (4)
Memory access time	3	15	(1)
Data transmission time	3	3	(1)
Time through Map Unit	5	5	(5)
Memory interchange time	3	3	(4)

Notes:

(1) Time may be hidden under previous map instructions.

(2) Time may be extended waiting for:

a) results to be available in the register file from load instructions or arithmetic computations;

b) a previous streaming instruction to enter the streaming instruction queues because of a flag conflict.

While b) holds up execution of map instructions, the time spent waiting is not counted against the startup time for the current instruction but rather against the instruction that caused the wait.

(3) Time may be extended waiting for:

a) read or write flags to clear that are in conflict with the keys of the current instruction;

b) instructions in the map queues to be issued to the Map Units.

(4) Time may be extended by the priority element holding the request or by the requested bank being busy.

(5) Time from a datum entering Map Unit until it leaves. Time may be extended by instruction and mode. For example, a compress using a very sparse control vector will give data a long time through the Map Unit.

3.12.4 Swap Unit Timing

Swap Unit startup consists of the issue cycle = 1, transmission to the Swap Unit = 1, and Swap Unit setup = 5 cycles. Once begun, the swap operation moves data at the rate of 512 data bits every 256 nanoseconds.

4.0 QUALITY ASSURANCE PROVISIONS - Not Applicable

5.0 PREPARATION FOR DELIVERY - Not Applicable

6.0 NOTES

6.1 Intercom

CDC FMP has an intercom system, utilized primarily for maintenance purposes, which can be enabled by simply plugging the required number of headsets into the desired intercom jacks. Jacks are located in each section and in the MCU. Up to four headsets may be on-line at any time.

6.2 System Startup

The Startup sequence for the system is as follows:

1) Bring up system power.

2) Autoload MCU.

3) Master clear the system from the CPU to:

    a) initialize the CPU - clear all control flip-flops, data flags, interrupts, and error flip-flops;

    b) set monitor mode in the CPU (Job Mode FF cleared in step A).

4) Load microcode into all FMP microcode memories from the MCU.

5) The MCU sends an external flag to the I/O stations required on-line. The stations, on receiving this flag, will autoload and enter an idle loop waiting for a channel flag from the CPU. An alternative approach is to manually autoload each of the stations desired on-line.

6) The MCU loads the operating system kernel into Intermediate Memory, forces the Map Units to transfer the kernel to Main Memory, and then interrupts the CPU. The CPU recognizes the interrupt and executes a partial exchange to start execution in monitor mode. This exchange is the same as a normal job to monitor exchange except the contents of the Register File are not stored. Program execution starts at the address contained in monitor's register six just as it does after a normal I/O interrupt.

APPENDIX A

UNIQUE SYNDROME WORDS FOR SINGLE BIT FAILURES

Bit	Data	Syndrome Word
0	80000000	70
1	40000000	68
2	20000000	58
3	10000000	64
4	08000000	54
5	04000000	7C
6	02000000	7A
7	01000000	76
8	00800000	1C
9	00400000	1A
10	00200000	16
11	00100000	19
12	00080000	15
13	00040000	1F
14	00020000	5E
15	00010000	5D
16	00008000	07
17	00004000	46
18	00002000	45
19	00001000	26
20	00000800	25
21	00000400	67
22	00000200	57
23	00000100	37
24	00000080	61
25	00000040	51
26	00000020	31
27	00000010	49
28	00000008	29
29	00000004	79
30	00000002	75
31	00000001	6D
32	Check Bit 0	40
33	Check Bit 1	20
34	Check Bit 2	10
35	Check Bit 3	08
36	Check Bit 4	04
37	Check Bit 5	02
38	Check Bit 6	01

(continued)

APPENDIX A (Cont.)

The syndrome word is latched if the bit shown in the data pattern in the above table is in error. For example, if and only if, bit 0 failed on any data pattern, then the syndrome word would be 70.

The SECDED error latching hardware has two basic modes of operation - Mode 1 and Mode 2.

Selection between the two modes is accomplished through the MCU/CPU Maintenance Line called SELECT SECDED ERROR LOG MODE TWO.

For both modes in the event of simultaneous SECDED errors, the information to be latched is dependent on the relative priority of the data buses or half-words which contain the errors. All information will be correct for the error selected. It is possible in both modes to encounter a single and double error simultaneously and latch the single error. The double error flag will set unconditionally. Therefore, if the double error flag is set, the syndrome bits must be checked to determine if single or double error was latched. In the event the single error flag is set, and no double error, the error will be a single error.

Mode 1

The first error to occur after a master clear or error clear will have its error information latched. The information will be correct in all cases, regardless of subsequent errors. If a double error follows a single without an error clear, the double error information will be lost.

Mode 2

Operation in Mode 2 is the same as in Mode 1 except for the following enhancement: An attempt will be made to latch the error information for the first double error encountered whether or not a single error has previously been latched.

As in Mode 1, the double error flag will set unconditionally when a double error is encountered. However, other aspects of Mode 2 operation are less certain. The conditions which may result are listed below:

Case 1

In the event of simultaneous errors, Mode 2 is the same as Mode

(continued)

APPENDIX A (Cont.)

1. If the double error flag is set, the syndrome bits must be checked to determine if a single or double error was latched.

Case 2

If the SECDED checker encounters a single or several single errors, and is absent of the double error flag, then the error information will be that of the first single error. All information is correct as in Mode 1.

Case 3

If the SECDED checker encounters a double followed by other double or single errors then the error information will be that of the first double error. All information is correct as in Mode 1. However, the MCU cannot be distinguished from Case 1 with the doubled error latched, so the syndrome bits must be checked.

Case 4

If the SECDED checker encounters a single error and N minor cycles later (N<8) a double error is encountered: Address bits 37 thru 54 for either the single or double error may be latched; bits 55 and 56 are indeterminate; and the remaining error information would be that of the double error.

Case 5

If the SECDED checker encounters a single error and N minor cycles later (N>8) a double error is encountered, the double error information will be correct. However, the MCU cannot distinguish this case from Case 4.

Case 6

If the SECDED checker encounters a double error and one or more minor cycles later a single or double error is encountered, this is simply Case 3. The first double error information will be latched.

Mode 2A Double Error Log

This mode is electronically identical to Mode 2. The difference is strictly operational. Specifically, after a master clear or error clear, the MCU deliberately creates a single error using

(continued)

APPENDIX A (Cont.)

the maintenance function to toggle a check bit. This error is not cleared, and effectively blocks detection of all subsequent single errors. Consequently, when the MCU detects the double error flag, it knows that this is Case 5 and the error log information is correct for that double error.

BLOCK WRITE ENABLES

The MCU has the capability to enable block write enable if a SECDED error occurs. There are two options which can be selected depending on SECDED error mode.

1. With Mode 1, the write enables will be blocked when SECDED receives its first single or double error.

2. With Mode 2, the write enable will be blocked when SECDED receives its first double error.

COMPLEMENT I/O CHECKWORD BITS

This maintenance feature enables the MCU to toggle the Write I/O checkword bits before write into memory. Toggling the 128 combinations on each half-word of the six Read Data Buses allows checkout of the SECDED checker.

GENERAL USAGE

Mode 1 is a good SECDED latch design for a memory with low error rate. All error log information is correct. However, it will not latch the double error if it follows a single error within the cycle time of the MCU.

Mode 2 is a better SECDED latch design for a memory with a high error rate. All single errors latched are correct, and all double errors following a single error by greater than eight minor cycles (80 ns) are correct. A double error occurring before a single error is also latched correctly.

Mode 2A is a double error logging system for use if single errors are to be ignored. This mode will miss the double error only if there is a simultaneous single error with higher latching priority. If this condition would occur, a diagnostic requesting only one bus will get around the bus priority. If the diagnostic fails and still latches a single error, then the double error is in a lower priority half-word.

APPENDIX B

ASYNCHRONOUS DATA MOVEMENT CONTROL FOR THE FLOW MODEL PROCESSOR

The movement of data to, from and within a vector processing element can be described by a Data Flow Model. This means that if a function is enabled, has data to process and a place to put the processed result, the data will be accepted and processed without specific external direction. The memory units (Main Memory and Intermediate Memory) act as an infinite source and sink of operands (tokens). The Map Units and the Vector Processor are the major processing elements, the Vector Processor performing as one element (node) and each Map Unit having several elements.

The idea is to provide each independent element enough control information so that a process can control itself and then, after completion of the process, can try to do something else. Because the individual elements are independent it is possible to have more than one process going on at the same time.

A process is defined as a chain of elements connected to each other is some serial fashion between a data source and a data sink (memory). Setting up a process involves three principle pieces of control information; the first is what function is to be performed by a processing element, the second is the source of data, and the third is its destination. Of course not all processing elements require all three pieces of control information: some elements will perform only one function, and others may be "hardwired" to another element thus obviating the need for connection information. But there will be no element, visible externally, that will not get at least one of the three pieces of control information.

There is, however, in the data movement control a major difference from a normal Data Flow Model. In such a model a processing element's (node's) output connection (link) must be empty before it will accept and process data. This is not acceptable in a pipelined computer because it implies that every other segment of the pipeline will be empty; i.e., only one-half the hardware is in use at any one time. In order to get greater use of the hardware a register is put after each element. The presence of this register means that a processing element can take input data even if the next element in line cannot accept data on this cycle because the segment has a place to put the result. If, however, the register is full and the next element cannot accept data, then processing must stop. Of course it is possible to look at the registers as an added set of actors so that the more classic model of data flow still holds.

(continued)

APPENDIX B (Cont.)

There are two control lines necessary between process nodes to facilitate data movement. The first provides notification to a successor node that a piece of data is available for it; this is called the "valid" line - a notification to the successor node that valid data is available. The successor node replies to the calling node with the second control line - the "accept" line, meaning that the node has accepted the data.

The introduction of the register solves a problem of hardware usage but still leaves one rather nonobvious problem. In a high speed computer the delay time of the signal paths is greater than the delay time going through the logical elements - and gates, latches, etc. This means that for processing elements which are physically separated, it is impossible for a processing element to tell the previous processing element that it has taken the data in time for the previous element to put new data on the links between the elements (in the same cycle). This can be fixed by placing multiple registers on the output of the first processing element to act as a data buffer between the distant processing elements. The buffer must be a FIFO (first in, first out) stack that is controlled by both processing elements. The number of words in the buffer depend on the electrical distance between the processing elements on each end of the buffer and on the reply time.

The FIFO is placed close to the sending element. As long as the FIFO is not full the FIFO will accept input valids (send accepts back to the sending element). The FIFO is full when all ranks of registers are full (holding data) except one. When the FIFO has any data (the FIFO is not empty) it sends a valid to the next element in line. Since the next element is some distance away it may be several cycles before the receiving element returns an accept. Because the input side of the FIFO keeps accepting data the FIFO will get filled to some depth before data starts moving at the FIFO output. After the first accept from the receiving element data will flow at the rate of one item per cycle into and out of the FIFO. Thus a certain percentage of the FIFO registers will remain full as long as both sending and receiving elements maintain the flow.

A process can then be redefined as a valid/accept chain from a source (possibly multiple sources) to a sink. This then is how the various vector processing elements are made to work together. The Scalar Unit, acting as the instruction issue unit, gives control information to the various elements necessary to perform some process. The elements use the control information to interconnect themselves and to control the movement of data without any further high-level intervention.

APPENDIX C

PDC HARDWARE DESCRIPTION

The Programmable Device Controller (PDC) is the principal element of a generalized I/O system that uses high-speed serial channels as interconnecting data paths between multiple processors and peripheral equipments. The generalized system, referred to as the Loosely Coupled Network (LCN), has been under development by Control Data for several years. The control of the serial channel can be distributed to all resident PDCs rather than being centralized in one network control processor.

Figure C-1 shows a block diagram of a PDC as configured for use in a generalized network trunk system. The PDC consists of five functional parts, four of which are common in the LCN environment and one of which is unique to the device or channel being interfaced. The trunk interface is a set of hardware and microcode that matches the PDC to a high-speed (50 megabits per second) serial trunk. The trunk interface is in turn comprised of two sets of logic, a trunk control unit interface (TCI) and from one through four trunk control units (TCUs). The number of TCUs that would be required depends on the number of different trunk lines present. The buffer memory is used for temporary data storage, allowing various devices with differing data rates to use the trunk. The processor controls the PDC resources and manages data flow. The device interface is a unique set of hardware that matches the device or processor channel to the PDC internal bus. The PDC design is such that common parts can be used with a multitude of unique processors or peripheral equipments. This commonality minimizes unique part types, reduces new unit design time, and most importantly assures a controlled serial trunk system structure.

PDC INTERNAL BUS

The PDC internal bus is used for inter-element communications. The bus consists of 16 data bits, two parity bits, 16 address bits, and several control bits. Bus usage is allocated equally among three elements: the trunk interface, the processor, and the device interface. Each element has a time slice approximately 106 nanoseconds wide which occurs once every 320 nanoseconds. Time slice allocation allows all three elements to access the bus, and therefore the memory, at a guaranteed 50 Mbps rate (16 data bits every 320 nanoseconds).

(continued)

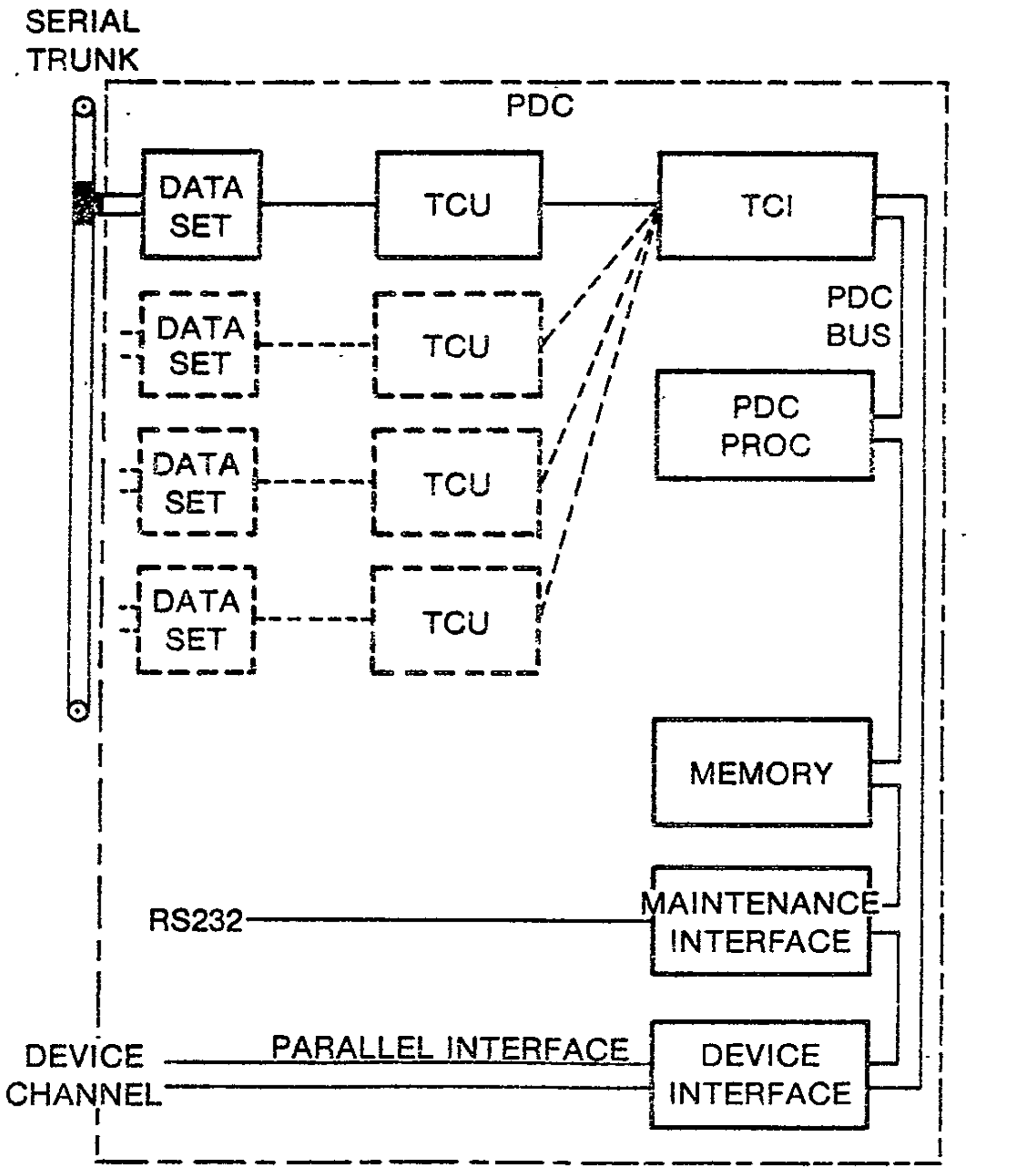


Figure C-1. PDC BLOCK DIAGRAM

APPENDIX C (Cont.)

PDC MEMORY

The PDC memory isolates the synchronous serial trunk and asynchronous connected devices, while buffering data rate differences between them. The size of the memory depends upon the attached device and the particular applications. Address lines allow a maximum configuration of 63,488 words with a word length of 16 bits. The memory cycle time is 106 nanoseconds.

PDC PROCESSOR

The PDC processor consists of hardware and controlware for the functions which manage the PDC resources and execute system functions. The processor normally performs the following tasks:

- Managing PDC resources such as allocation of buffer memory and reservation of the PDC for a particular message source.
- Handling message flow which includes generating message headers, monitoring data transfers, and interpreting received messages.
- Initiating error recovery procedures on serial channel transmission errors as well as device errors.
- Executing various system functions which may include queuing processes, executing I/O processes through the serial channel and device interfaces, handling device drivers, translating message formats if required, and generating autoload messages if required.

The PDC processor is a 16-bit miniprocessor constructed of 4-bit microprocessor chips with a microcode instruction implementation. The microcode memory runs at a cycle time of 160 nanoseconds. The number of microcode references per processor instruction varies depending upon the instruction being executed; however, the average is approximately six to eight microcode references. This gives an average processor instruction time of 960 to 1280 nanoseconds.

(continued)

APPENDIX C (Cont.)

DEVICE INTERFACE

The device interface electrically adapts a device channel to a PDC. Other key functions that may be performed by the device interface include:

- Transfer of data and commands between an attached device and the PDC.
- Assembly/Disassembly to handle different word sizes.
- Device control for passive devices like disk and tape.

The device interface is that set of logic which is unique to a PDC, depending on the device being interfaced. Currently scheculed device interfaces include CY170, CY205, IBM 370, DEC 11, 844/FMD, CY18, and 819. Other device interfaces are currently being analyzed and will be developed as appropriate. With the exception of the DEC 11, all the device interfaces include a single channel connection between the PDC and the device. The DEC 11 PDC will contain a single channel interface as standard, with options to connect up to four DEC 11's via a single PDC.

TRUNK INTERFACE

The trunk interface performs the following functions when the PDC is used in an LCN system:

- Interface the data set.
- Add/Delete the serial trunk protocol envelope which includes cyclic redundancy code (CRC) generation and detection.
- Interpret message functions and react accordingly.
- Generate response messages to ensure closure for all valid incoming messages (some form of response will always be generated even if the PDC processor or attached device is unavailable).
- Access the trunk if the PDC needs to send a message.

(continued)

APPENDIX C (Cont.)

The trunk interface can operate in two modes: message mode and streaming mode. Message mode is the normal mode of operation in which all transactions consist of a command and response message pair. At the end of the message pair, control of the serial trunk is relinquished to allow other PDCs on the trunk to use the trunk. Streaming mode is used in special cases where a high rate is required. In streaming mode, the trunk is held between message/response pairs, allowing the next message to be sent as soon as it is ready.

Access for the trunk usage is resolved by a rotating priority mechanism. This method guarantees trunk access to all units on the trunk during peak loading of message mode traffic.

The standard trunk interface is comprised of a Trunk Control Unit Interface (TCI) and a Trunk Control Unit (TCU). Additional TCUs can be field installed to permit access to up to four different trunks.

SERIAL TRUNK HARDWARE

A 50 megabit data set and coaxial transmission media are used for the serial trunk. The data set uses a phase modulated carrier system to transmit data in a synchronous burst mode. High quality coax cable and type-F connectors are used to eliminate possible ground and EMI/RFI problems.

The performance objectives for the data set and tranmsimssion media are:

- 50-Mbit/s transmission rate.
- Trunk length maximum of 1000 feet with 16 attachments.
- More than 16 attachments are possible with restricted trunk lengths.
- Longer trunk lengths (greater than 1000 feet) are possible with fewer attachments and/or higher quality cable.

A 16-bit cyclic redundancy code (CRC) is inlcuded in every message frame transmitted across the serial trunk. The CRC is an extremely powerful error detection mechanism in that single-bit and multiple-bit errors anywhere in the message frame are detected.

APPENDIX D

FMP FUNCTIONAL AND TIMING CHARACTERISTICS

D1.0 Introduction

The CDC FMP consists of twelve functional elements when viewed at the top level of block definition. These are:

- Scalar Unit
- Streaming Control Unit
- Vector Streaming Unit
- Vector Unit
- Memory Interchange
- Main Memory
- Main Map Unit
- Intermediate Map Unit
- Intermediate Memory
- I/O Unit
- Swap Unit
- Backing Store

Each of these has a unique set of timing characteristics for its internal functions and for its interface to other elements. For a given operation or instruction, several of the elements may be involved, serially, and/or in parallel. For this reason, the timing characteristics which follow are presented by unit, or element, and timing for a given operation can be determined based on which elements are involved.

It should be noted, however, that not all twelve functional elements appear as individual sections.

- Vector Streaming Unit and Vector Unit are combined and described as Vector Operations.
- Memory Interchange is included with the discussion of Main Memory Access.

(continued)

D1.0 (Cont.)

- Main Map Unit and Intermediate Map Unit are presented together since, for some operations, they can operate as one unit.

- Swap Unit and Backing Store are one form of I/O and are therefore included under the discussion of Input/Output.

This results in the following categories of timing characteristics being presented.

- Instruction Issue (Scalar Unit)
- Main Memory
- Streaming Control
- Vector Operations
- Map Units
- Intermediate Memory
- Input/Output

D2.0 Instruction Issue

Instructions are issued by the Scalar Unit when they meet the proper requirements for issue:

1. Instructions are present to be issued.

2. Issue is not prohibited by some previous instruction. For example, issue is prohibited during branch sequences and is also prohibited for the duration of a SWAP (7D) instruction.

3. The registers referenced in the Register File by an instruction to be issued are not reserved, thereby conflicting with this instruction. Under certain conditions, this conflict is avoided for some scalar instructions that can make use of the results of the scalar floating point directly (see 3.2.6).

4. For vector streaming instructions (vector and map), no dependency or interlock key conflict exists from a previous streaming instruction.

(continued)

D2.0 (Cont.)

5. For vector streaming instructions, the instruction buffer for that instruction is not full.

6. For scalar instructions which return results to the Register File, the write data will not cause a conflict with some previously scheduled write in use of the single write port; e.g., if the instruction to be issued intends to return a result to the Register File in four cycles and an instruction that issued two cycles prior will return a result six cycles from when it was issued, then the issue of the current instruction must be held up for one cycle.

7. For main memory load/store instructions and for intermediate memory load/store instructions, the respective instruction buffer is not full.

Issue as used here means that the issue control portion of the Scalar Unit has passed control of an issued instruction to some other functional element. This does not mean that the instruction is lost, for a copy is kept in the instruction buffer for some time, or that the operation commanded by the instruction is completed. Issue means only that the resources required by the instruction are available and that the instruction is now free to run to completion.

The number of cycles it takes to issue an instruction, if of course issue is not blocked for some reason, depends on the number of cycles it will take to get required data from the Register File. The Register File can perform two separate reads every clock cycle. Most instructions, since they require only one or two source operands, will therefore issue in one cycle. Those instructions that require more that two reads from the Register File will take more than one cycle to issue. Thus a scalar store instruction which requires three reads from the Register File (a base address, an index, and the store data) requires two cycles to issue under ordinary circumstances.

D2.1 Issue Timing Parameters

The expected timing parameters for issue are:

- 15 cycles from request for an instruction from memory to receipt at the instruction stack (as the result of a branch, for example). Since the machine performs instruction look ahead most fetch time is hidden.

(continued)

D2.1 (Cont.)

- 10 cycles for a branch not taken or for a branch in stack.
- Five cycles from the time an instruction is requested from the instruction stack until it is ready to be issued. Again this time is normally hidden because there is an issue pipeline that enables the machine to issue every clock cycle unless held up for some reason. It is while an instruction is in the issue pipe that the various things that can hold up issue of a particular instruction are checked.

The timing of the various instructions may be found in section 3.12.

D3.0 Main Memory

The FMP memory is physically 64 memory modules, with each module containing eight memory banks of 16,384 64-bit words plus SECDED (a total of 8,388,608 words, 131,072 in each module). Thus one request sent to each module will get 64 64-bit words, or a total of 4,096 data bits. In order to reduce the total number of wires, memory is organized into four accesses of 16 modules, each access having a total of 1,024 bits. Each access is independent so that memory can support up to four separate memory requests simultaneously.

Each request to a memory access, whether for a read or a write, will make a bank busy in all 16 modules for three clock cycles.

Memory is addressed first across the modules and secondly down the banks. Thus, address 0 is on module 0, bank 0; address 1 is module 1, bank 0; address 64 is module 0, bank 1; etc. This means that a 1,024-bit fetch from memory will get 16 consecutive 64-bit words (or 32 consecutive 32-bit words). Any memory request is for 1,024 bits, whether for read or write.

Each bank within a module is independent of the other banks of the module except that the address and data lines are shared. Thus an access can receive a new request each clock cycle as long as the request is to a bank that is not busy (the requested bank has not had a request in the previous two cycles).

Functionally, then, the eight million word Main Memory is four 1,024-bit accesses, each having eight banks.

D3.1 Main Memory Access

Since the Vector Unit uses 512 bits per clock cycle per input stream, on the average a stream will make a memory request every other cycle. Simplified memory access control is facilitated because of the knowledge that a vector stream will make a memory access request every other cycle and that the address will be for the next sequential 1,024-bit access. Because it is expected that vector performance will be the limiting factor on overall machine performance, one of the principle jobs of the Memory Interchange is to optimize vector streaming memory access requests. The interchange does this in two ways.

1. If there are two or more requests that conflict for memory on a particular cycle, the internal priority control will grant access first to a streaming request.

2. If an access is in streaming mode, the priority control will prevent access to the banks that the streaming access will require next if that conflicting access will slow or stop the streaming access; i.e., banks are reserved prior to use.

As an example of the above consider:

(a) A vector read request at some clock cycle n for bank 0 of modules 0 through 15.

(b) A map READ1 request on the same cycle for one 64-bit word in bank 0 of modules 0 through 15.

(c) A scalar load/store request at time n+2 for one 64-bit word from bank 0 of modules 16 through 31.

Requests (b) and (c) will both be held up by the streaming request (a). Request (b) will be granted at time n+3 and request (c) will be granted at n+5. Note that both request (b) and the second request from request (a) will be granted at n+4 because the requests will be to nonconflicting banks. Note also that if the load/store request had come at n+1 the request would have been granted because bank 0 of modules 32 through 47 would then be busy from n+2 through n+4 thus being completed so that the streaming request could be granted at n+4 to make the banks busy from n+5 through n+7.

D3.2 Main Memory Access Timing Parameters

The expected access timing parameters are thus a three cycle bank busy with the busy timing starting one cycle after the grant of access and data being available on the last cycle of the bank busy when reading data. Note that access to a given bank can be granted during the last cycle of a three cycle bank busy provided no conflict exists beyond that bank busy.

D4.0 Streaming Control

The principle function of the Streaming Control Unit is to receive all streaming instructions (map or vector) issued by the Scalar Unit and route appropriate setup data to the unit or units which will execute them in such a manner as to avoid conflicts of data in memory and/or to prevent data and instructions from getting out of order. This is accomplished by read keys and write keys which are fields of the streaming instructions, a dependency flag register associated with Main Memory, and an interlock flag register associated with Intermediate Memory.

Each flag in the dependency flag register consists of three bits as follows:

Bit 1 - a read reference with this key by the Vector Streaming Unit;

Bit 2 - a read reference with this key by the Main Map Unit;

Bit 3 - a write reference with this key by either the Main Map Unit or the Vector Streaming Unit.

The interlock flag register is a single bit per flag, and any read or write reference to Intermediate Memory with a given key will set that respective bit.

For vector operations read and write keys are both applied

(continued)

D4.0 (Cont.)

against the dependency flag register since Main Memory is always source and destination for these operations. Map operations, however, can have either Main Memory or Intermediate Memory as source and/or destination, and can therefore have read or write keys applied against the dependency flag register or the interlock flag register as follows:

> If the source (read) is the Intermediate Memory, the read key will be applied against the interlock flag register. If the destination (write) is the Intermediate Memory, the write key will be applied against the interlock flag register. If both the source and destination are Intermediate Memory, two interlock flags may be active during that operation. Keys that do not apply to the Intermediate Memory will be applied against the dependency flag register (main memory read or main memory write).

For any operation a key of zero means no flag, i.e., no dependencies exist, no checks are made, and no flags are set for that operation. Note that it is possible, however, to have a read key or write key zero and the other non-zero.

When a streaming (map or vector) instruction is decoded by issue, the requisite data is sent to a set of registers in the front of the Streaming Control Unit. The keys are checked, as required by the instruction. If there is a key conflict the instruction will be held in the registers and a flag will be sent back to prevent issue of additional streaming instructions.

- A read key of an instruction will hold up the instruction from entering the execution queue if either the write bit of the requested flag is set, or the read bit of the requested flag for the same functional unit is set; e.g., a <u>vector</u> instruction read key checks the vector read bit for the requested flag.

- A write key of an instruction will hold up the instruction from entering the execution queue if either the write bit of the requested flag is set, or the read bit of the requested flag for the opposite functional unit is set; e.g., a <u>vector</u> instruction write key checks the map read bit for the requested flag.

- When the instruction is ready to enter the execution queue, the proper read bit of the corresponding flag will be set, and the write bit will be set according to the key in the write key field.

(continued)

D4.0 (Cont.)

- The Vector Streaming Unit will signal the Streaming Control Unit to clear the vector read bit when the read ports have finished reading the data for the instruction. The same will be done for the vector write bit when the write ports have written the last data for the instruction into Main Memory.

- The Map Unit will signal the Streaming Control Unit to clear both the map read bit and the map write bit when the last data for the instruction has been written into Main Memory since map instructions have no input (read) length. (More detailed design may reveal that the read bit may be cleared a few cycles earlier than the write bit in anticipation of a completed write.)

Note that this does not prevent the issue unit from continuing issue unless the next instruction to be issued is another streaming instruction.

The flags corresponding to the read and/or write keys are checked to see if the flags are clear, the flags become clear, or the flags are not checked (key code=0). When this is true the instructions wll enter the proper instruction queue(s) and set the proper flag(s) as required. If the instruction is a map function the control bits of the instruction are checked to see if the instruction is to run in the Main Map Unit, the Intermediate Map Unit, or both. If the instruction is to run in one unit only, it is sent to the respective queue. If the instruction is to use both units, the proper parts of the instruction are put into each map queue and a tag is appended to the queue entry. As the instructions are about to exit the queues, they are checked for a tag. If a tag is found in one queue, that queue stops and waits for the same tag in the other queue. When both tags are available, the data in both queues are sent to their respective map units.

When the last data for a streaming instruction is on its way to memory such that no possible out-of-order sequence can occur, the key number is sent back to the Streaming Control Unit to clear the flag associated with that instruction.

D4.1 Interlock Flags

Three functional units, the Intermediate Map, I/O, and Swap Units, are aware of the interlock flags. Only the Intermediate Map Unit receives its key information through the Streaming

(continued)

D4.1 (Cont.)

Control Unit. Both Swap and I/O receive the key information as part of control messages that are left in Intermediate Memory. The Scalar Unit will write a message, e.g. to the Swap Unit, in Intermediate Memory and then send an alert to the Swap Unit. The Swap Unit will decode the message and attempt to execute the required commands. If the command requires the movement of data (most messages will), the Swap Unit will send to the Streaming Control Unit an inquiry as to the state of the flag corresponding to the particular key specified as part of the control message. Until the Streaming Control Unit replies 'flag clear' the Swap Unit (or whatever) can not proceed. If (when) the flag is clear, the Streaming Control Unit will set the flag as an interlock on Intermediate Memory. At the completion of the command(s) the unit involved will send the key back to the Streaming Control Unit again with a command to clear the required flag.

D4.2 Keys and the Data Flag Branch Registers

Scalar load and store instructions, both to Main Memory and to Intermediate Memory, do not have dependency and interlock keys. To enable a running program to check the progress of a sequence so as to ascertain whether a required result is available in memory, the dependency flags and the interlock flags are available in the Data Flag Branch (DFB) Register. This information can be used by the programmer in two ways: either the programmer can explicitly test for a particular flag, or the DFB Register can be conditioned so as to cause a program interrupt when a particular flag (or flags) goes clear. See the functional and instruction specifications for use of the DFB Register.

D4.3 Streaming Control Unit Timing Parameters

If no key confict exists for an instruction and the appropriate instruction queue is clear, three cycles elapse from the input to the Streaming Control Unit until the instruction exits to be executed.

D5.0 Vector Operations

A major problem in pipelined vector computers, particularly when running short vectors, is that which is known as 'vector startup' time. This is the time from issue of a vector

(continued)

D5.0. (Cont.)

instruction until the first data is available; in the FMP this is when the first result is returned to memory. The effect of this startup time is to reduce the efficiency of the machine when executing short vectors. (It might take, for example, 20 cycles to get one vector result and 120 cycles to get 200 results). The problem has been attacked, in the FMP, by spreading out the control functions among several elements. Each separate element is then free to process its own piece of the problem. This means that instruction execution is much more parallel than previously done. So much so, that for very short vectors each functional element can be working on a different vector.

D5.1 Vector Read Port

The first functional element in performing a vector operation is a vector read port. The read port receives the memory address and also the number of elements to be read by this vector stream (input length) and the number of elements to be written by this instruction (output length). There is no requirement that the two lengths must match. If the input length is shorter than the output length there are parameters in the instruction being executed which specify how to continue. If the output length is shorter than the input length, when the output length is satisfied any unsatisfied requests for data from memory are abandoned. Note that when the output length count in the read port goes to zero the port starts another instruction even though the data from the read stream just completed has not yet reached the vector pipelines for execution. The pipelines may not even know that data is coming.

Another function of the read port is physical operand alignment. A vector may start on any 32-bit memory boundary (64-bit boundary for a 64-bit data stream). Because memory requests are always for 1,024 bits on fixed bank boundaries, the data may need to be shifted so as to get corresponding operands from different streams together. This means that it is likely that some of the data on the first fetch is not part of the data specified by the instruction. This 'extra' data is lost when the data is aligned. In order to simplify control, the first operand in each stream is always sent to pipe 1.

As data is available at the output of the read port it is passed to a FIFO (first in, first out) buffer and a flag is generated which tells the FIFO element that fact. The FIFO will accept

(continued)

D5.1 (Cont.)

data until it becomes full.

When the input length count goes to zero, the read port attaches a flag to the last data as it leaves the read port. This flag is used only in the case of recursive instructions (Sum, Dot Product for example) where the instruction length is defined by an input length count instead of an output length count.

D5.2 FIFO Buffer

The second functional element in the vector data path is the FIFO buffer which facilitates temporal alignment of streams while maintaining data streaming. The vector pipelines require data to be available at specific times with respect to other data. A data stream arriving from Main Memory may not be timed properly with respect to another data stream. Causes for this, in Main Memory, are bank busy conflicts and priority conflicts (a request for memory is delayed to allow another request to proceed first). The FIFO thus provides a place to put data that has arrived ahead of other data, but not stop streaming the earlier data while waiting. (If a data stream must stop making memory requests because there is no place to put the resulting data, the streaming slot in the Memory Interchange is lost. Several clock cycles are then lost in starting the data stream again).

Another similar use for the FIFO buffer is required by the vector pipelines. If the operation (A+B)*(C+D), for example, is performed, all four input streams are required at the same time. But if the operation is A*(B+C*D) instead, then the stream carrying the B data must be delayed with respect to the C and D streams. In a like manner, the A stream will wait until (B+C*D) is available before sending its operands to the pipelines.

D5.3 Pipelines

The next element in the vector data path is the pipeline switch which is not visible to the programmer; the principle data function is that at this point the eight 64-bit words of a data stream are multiplexed into four pipelines (Vector Units). It is also encountered again at the output of the Vector Units where the principle function is to de-multiplex four pipelines to eight words of a data stream. Time through the pipeline switch is included in the pipeline timing below.

(continued)

D5.3 (Cont.)

From a timing standpoint the Vector Unit is a variable delay line. The amount of delay depends on the function being performed. (Delay, as defined here, is the amount of time from the entry of the first operand of the first data stream until the first result is available at the pipeline output.)

D5.4 Vector Write Port

The final element in the vector operation path is the vector write port. It functions in a manner very similar to the read ports. The major difference is that the shift functions for operand alignment work in reverse: the pipe 1 result is shifted to place the result in the proper place in the 512-bit result stream; all other pipeline results are then shifted by a like amount.

D5.5 Vector Operation Timing Parameters

- Two cycles read port setup time (This time can be hidden between instructions if another instruction is awaiting execution).
- Eight cycles from read port request to Main Memory until the data is received (assuming no bank busy conflicts in memory).
- Two cycles through the read port to the FIFO.
- If the FIFO is empty and the pipeline is willing to take data, the transit time of the FIFO is one clock cycle.
- The current expectation for pipeline timing for the non-recursive instructions is as follows:

Cycle Delay	Operation
9	Add, Multiply, Multiply-Add
12	Add-Multiply or Add-Multiply-Add
15	Other operations such as Multiply-Add-Multiply
18	Second Divide Pass (64-bit result)

(continued)

D5.5 (Cont.)

Cycle Delay	Operation
20	First (or only) Divide Pass

The timing for recursive operations (Sum, Dot Product, etc.) is very complex; a timing algorithm and general timing information will be supplied later.

- Two cycles through the write port.
- Three cycles from exit of the write port until the data is being stored in memory (if no bank busy, etc).

D6.0 Map Units

The two map units - the Intermediate Map Unit (IMU) and the Main Map Unit (MMU) - can each function separately or they can work together. Both map units have the following functions when working independently:

GATHER
SCATTER
COMPRESS
MASK
MERGE

These functions are performed memory to memory, each unit using its own memory. See section 3.5 for a description of each function.

The map units working together can perform the following functions:

GATHER - Intermediate to Main Memory
COMPRESS - Intermediate to Main Memory
SCATTER - Main to Intermediate Memory

The functions performed are the same as the functions performed by the units working independently.

D6.1 Main Map Unit

The timing of the Main Map Unit is fairly complex. Two primary difficulties are encountered when attempting to determine map unit timing:

(continued)

D6.1 (Cont.)

1) The Map Unit has second priority on memory requests; thus requests can be held up for variable amounts of time.

2) The Map Unit is seldom able to 'stream' its memory requests. In other words, it may not know ahead of time when or where the next memory request will be made - memory is 'catch as catch can', i.e., requests are relatively random in time and/or address.

The first difficulty is extremely variable. If the Vector Unit is waiting for completion of a map function (through the dependency flags), the Main Map Unit will have Main Memory virtually all to itself. If however, the Vector Unit is running, from 3/8 to 3/4 of the memory bandwidth will be tied up by the Vector Unit. If it turns out, in general, that the map to memory access is a bottle neck the Map Unit may be given access priority over the Vector Unit; at the present stage of design analysis, this is considered not to be required.

The second difficulty has several facets.

- In a GATHER or SCATTER operation, the starting address for any record loaded (GATHER) or stored (SCATTER) is, of course, not fixed on the memory access boundaries (at least 1K bits/fetch on 1K boundaries). This means that some of the data, on at least a good portion of memory requests, will not be used. The effective bandwidth of the function is thus reduced. As a worst case example of this, take a GATHER operation in 32-bit mode with the starting address of a 2-word record being on the last 32 bits of a 1024-bit memory bank. This means that 2K bits must be fetched (taking from three to eight cycles at the memory access) to give 64 resultant data bits.

- In the COMPRESS operation the output rate is controlled by the density of permissive bits in the control vector.

- In the MERGE operation the rate of movement of the input streams is dependent on the control vector.

All the above cases mean that for some streams, although it may be known where to put or get the next data, the time may not be known.

The MASK operation is an anomaly to the above difficulties. All three data streams (two inputs, one output) can move at streaming rate. But all is not roses; if the Vector Unit is using all its streams (six), then the Map Unit comes up one

(continued)

D6.1 (Cont.)

short because memory allows only eight simultaneous accesses to memory. In this case, the MASK operation will run about half streaming rate because no map streaming will be possible.

In order to get around some of the hardships on the GATHER operation (the most used map function) the Map Unit will attempt to get two records at a time. In the case of long records, where the input has a chance to stream, this is very little advantage. For short records (say ≤ four words) the use of two read steams will help a great deal - as much as 100 percent improvement in throughput.

The MERGE is a very complex operation from a logician's point of view, so much so that an eight-way parallel merge is about all that is practically possible. This means that only 8 operands are processed during any one cycle. The operand size does not make any difference to this operation, with respect to the data rate. This is different from other instructions or operations which move twice as many 32-bit data words/cycle as 64-bit data words. This restriction is also true of the COMPRESS operation.

D6.2 Intermediate Map Unit

The Intermediate Map Unit can perform the same operations as the Main Map Unit but at reduced performance since it is operating with a lower performance memory.

The IMU is connected to Intermediate Memory by three 256-bit (plus SECDED) ports. Each port, if utilized fully, is capable of moving a 256-bit quantity every three clock cycles (48 ns). The nominal throughput of an intermediate memory to memory operation is thus one-sixth of the Main Map Unit performing main memory to memory operations.

There are, however, variables that will affect this rate:

- Intermediate Memory will have less contention for memory bandwidth.
- The present Intermediate Memory design is 'slightly' block oriented (while the memory appears to have perfectly random access to the programmer, requests for less than 32 64-bit words may be slowed down from the full streaming rate).
- MASK and MERGE operations require four data streams (two data inputs, one data output, one control vector input).

(continued)

D6.2 (Cont.)

Since the Intermediate Memory has only three ports available to the IMU, one port must be timeshared to provide the control vector data. This slows down the throughput by about three to five percent for those operations from what would otherwise be expected.

D6.3 Combined Map Unit Operations

When the two map units work together to get data from/to Intermediate Memory to/from Main Memory, two ports of the Intermediate Memory are tied together. This means that 512 bits will be available every three cycles instead of 256 bits otherwise. The two ports will work together on the same record of a GATHER or SCATTER for long records (> 32 64-bit words). The two ports will process alternate records for shorter record lengths. On a COMPRESS the second port is set to a fixed address amount (32 64-bit word) ahead of the first port and the address of each port is incremented by twice the normal count; that is, the ports get alternate 32-word groups.

D6.4 Map Unit Timing Parameters

Timing parameters for the map units are as follows:

- Main memory access time is the same as that for the Vector Unit but with greater chance of conflicts.
- Intermediate memory access time is about 20 cycles (memory itself, being dynamic MOS, will have a cycle time of about 24 cycles).
- Three cycles, best case, for data to move through either map unit operating independently.
- Four cycles, best case, for data to move through both map units when operating combined.

D7.0 Intermediate Memory

The Intermediate Memory consists of 33,554,432 64-bit words of random access memory. It is accessed through four high speed ports and up to eight low speed ports.

D7.1 Organization and Access

The Intermediate Memory is organized as four memory groups with each group having four memory banks and each bank having eight memory modules. Each module has 262,144 72-bit words (64 data bits plus eight SECDED bits). The four memory banks in each group are driven in parallel; that is, the same address request is sent to each bank in the group at the same time. Thus, data is available at the output of the memory group 288 bits wide (256 data plus 32 SECDED).

When a memory group is accessed, the memory control interprets the lower three bits of the starting address to determine an initial module number. That module set (four modules in parallel) will be accessed, and every 48 nanoseconds thereafter the next successive module set will be accessed through the remainder of the 32-word block. Thus, a request to a group will get a variable amount of data. If the three bits of the request address are 000, then eight 288-bit data transfers will result. If the lower bits of a request address are 101, then three 288-bit transfers will result (module groups 5, 6, 7). The cycle time of a memory group is 384 nanoseconds (24 FMP clock cyles).

Because of the anomaly caused by the modules of a group not starting simultaneously, throughput to memory is maximized by starting requests at module set 0 (lower address bits 000) as much as possible, and by making as much use as possible of the 32 words thus transferred. The port controls of the Map, I/O, and Swap Units have this built into them.

D7.2 High Speed Ports

Each of the four high speed ports can move data at a rate of 288 bits every 48 nanoseconds. Each port has a small buffer to hold up to 32 words in case the requested memory group is busy or the port is denied access because of priority.

D7.3 Low Speed Ports

The low speed ports are made into two sets of four ports. Each set appears to the memory control as a high speed port. Thus to memory control, the memory is accessed by six high speed ports. Each low speed port can move data into and out of the port at a rate of 17 bits (16 data plus one parity) every 96 nanoseconds. Each low speed port also has a 32-word, 64-bit buffer. If a write operation to Intermediate Memory does not terminate on a 64-bit boundary the rest of the 64-bit word is

(continued)

D7.3 (Cont.)

zero filled when the data is written. For this reason the I/O Unit and Swap Unit perform a read-modify-write for a write operation which does not terminate on a 64-bit boundary, thereby avoiding the zero fill.

D8.0 Input/Output

The input/output of the FMP consists of two major subsections - the I/O Channels and the Swap Unit. They are described here together because to the system the Swap Unit is controlled identically with the I/O Channels; that is, to the FMP the Swap Unit _is_ an I/O Channel.

D8.1 I/O Channels

The FMP will support any number of I/O channels up to 14 with typical numbers being about six to eight. Each channel consists of one PDC (Programmable Device Controller) connected to the Support Processing System and/or the Disk Storage Subsystem through a network of serial data trunks. Each data trunk is capable of moving up to 50 million bits per second. A control protocol is used among all the PDCs connected to a trunk which prevents any particular PDC from dominating the trunk (possibly as a result of a failure) and allows all PDCs connected to the trunk some share of the trunk's time. Each PDC may be connected to one through four serial trunks.

Each PDC consists of four major elements:

1. Trunk control/interface unit - this element connects the PDC to the serial data trunk.

2. Device interface - this element connects the PDC to the FMP.

3. Processor - The processor is the intelligence of the PDC; it is aware of (and controls) what is going on in both interfaces.

4. Memory - This memory holds both the instructions and execution tables for the processor, and provides data buffers for data going through the PDC; the memory size can vary between 8K and 64K words depending on requirements.

The processor program code can be downline loaded into the memory either through the serial trunk or through the device interface, or it may be kept permanently in read-only memory.

(continued)

D8.1 (Cont.)

The PDCs are connected in pairs to the low speed ports of Intermediate Memory. Each PDC can move data at the rate of 16 bits every 320 nanoseconds (50 MHz bit rate). Each low speed port can take data at the rate of 16 bits every 96 nanoseconds (167 MHz bit rate); therefore, two PDCs connected to a low speed port can time share the port with very little conflict. Each PDC puts (takes) its data into (from) a 256-word buffer. The port control logic is aware of the amount of data in the buffer, and when the buffer is half full, an Intermediate Memory request is made.

The Intermediate Memory moves data at its highest data rate if the amount of data to be read or written is 32 64-bit words (or a multiple thereof) and the starting address is zero, modulo 32. The port logic takes special action if the starting address is otherwise, or if the amount of data left in the buffer is not a full set of 64-bit words at the end of a write to Intermediate Memory. On a first write to Intermediate Memory the port will make a memory request when the data to be written crosses a 32-word boundary, and will make memory requests thereafter for each 32 64-bit words accumulated in the buffer (buffer is half full).

On the last write of a transfer the port control will look to see if an integer number of 64-bit words are remaining in the data buffer. If so, the data is written into Intermediate Memory. If the data remaining is not an integer number of 64-bit words, the port will generate a read of Intermediate Memory and combine the read and write data to fill out the data to be written back (read-modify-write). On read operations extra data requested by the port but not used by the PDCs are discarded.

The I/O Channels do not receive control directly from the Scalar Unit. Instead, control messages are placed (stored) in the Intermediate Memory for the individual I/O Channels. The channels do not poll Intermediate Memory but instead are given an "alert" call by the Scalar Unit.

An initial intermediate memory address for receiving control messages is built into each PDC and this address is used to fetch the first message. One of the parameters stored in the control message is the location that will contain the next message.

The I/O Channels also leave status information in Intermediate Memory in a similar manner.

(continued)

D8.1 (Cont.)

The I/O Channels will have access to all of Intermediate Memory and will move all I/O data in blocks of 32,768 64-bit words.

D8.2 Swap Unit

The Swap Unit is connected to Intermediate Memory through two different ports; data flows through high speed port 3 and command information through low speed port 7. This allows the Swap Unit to be controlled, by the system, in a manner identical to the I/O Channels.

Data in the Backing Store is organized into 32,768-word blocks allowing the Swap Unit to appear as a disk to the FMP. This, of course, is of great benefit to the system software (it also means that, with proper software, the Backing Store can be an option).

The Backing Store can move data at a rate of 512 bits plus SECDED every 256 nanoseconds (16 clock cycles). A high speed port can move 512 bits plus SECDED in 96 nanoseconds (256 plus SECDED every 48 nanoseconds). The extra bandwidth is available to the Scalar Unit for access to Intermediate Memory.

The access time of the Backing Store is about 200 nanoseconds even though the Backing Store employs serial memory devices. This is accomplished by starting a block transfer at the current address within the block at the time of request, continuing to the end of the block, and starting then at the head of the block until the initial address reoccurs. The "access" time of a normal serial memory is virtually eliminated.

D8.3 Scalar Unit Access to Intermediate Memory

The Scalar Unit timeshares high speed port 3 to the Intermediate Memory with the Backing Store which uses about 40 percent of the available bandwidth, leaving ample resources for the Scalar Unit.

When scalar unit data is to be written to Intermediate Memory, a small buffer accumulates words into groups of 32 words (64-bit words). If the data crosses a 32-word boundary, that data up to the boundary is written to memory. If a 32-word group is only partially filled at termination of a write operation, a partial write is performed. If the last data to be written to Intermediate Memory does not end on a 64-bit boundary (32-bit

(continued)

D8.3 (Cont.)

writes are permitted), data is fetched from Intermediate Memory to fill the last word before the partial write is performed.

The minimum time scalar data will reside in the Swap Unit port is about three clock cycles. Throughput will be decreased if:

1. the Swap Unit is using the port (hold from 48 to 384 nanoseconds);

2. the data is being accumulated (see above);

3. the port is denied access (see section D7, Intermediate Memory).

APPENDIX E

CHECKING FUNCTIONS DURING PIPELINE PROCESSING

The following table summarizes the checking of functional elements during the various suboperations executed by the vector ensemble.

The following notes apply:

a) The functions performed for each suboperation code may be found in the FMP instruction specification found in Division 2 of this Volume.

b) AR2 used, AR2 not used refer to whether the AR2 result is to be stored in Main Memory or not.

c) A, B, C, D in the column headings refer to checkers A, B, C, and D which check the frontend adders, multiply, backend adders, and the complement, respectively.

d) The numbers in each column mean that checking is active during the indicated suboperation as follows:

   1 Network being checked but function is not used by this suboperation.

   2 Function is used by this suboperation.

   3 Checking is of final result.

e) A blank in the table means no checking is performed.

f) AR2 results are never used in 30-36 suboperations.

(continued)

APPENDIX E (Cont.)

Suboperation	AR2 used A	AR2 used B	AR2 used C	AR2 used D	AR2 not used A	AR2 not used B	AR2 not used C	AR2 not used D
00	1			1	1	2	3	1
01	1			1	1	2	3	1
02					1	2	3	2
03					1	2	3	2
04					1	2	3	1
05			3	1			3	1
06				1				1
07		2	3	1		2	3	1
08		2				2	3	2
09								
0A			3	1			3	1
0B			3	1			3	1
0C		2				2	3	
0D				1			3	1
0E								
0F				1				1
10						2		
11						2		
12							3	
13								
14								
16					1	2	3	2
17					1	2	3	2
18					1	2	3	2
19					1	2	3	2
1A					1	2	3	2
1B					1	2	3	2
1C					1	2	3	2
1D					1	2	3	2
1E	1			1	1	2	3	1
1F	1			1	1	2	3	1
20								
21								
22								
23								
24				1				1
25				1				1
30					1		3	1
31					2		3	1
32					1		3	1
33					1	2	3	1
34					2	2	3	1
35					1		3	1
36					1		3	1

DIVISION 2

FMP INSTRUCTION SPECIFICATION

CDC[R] FLOW MODEL PROCESSOR

INSTRUCTION DESCRIPTIONS

TABLE OF CONTENTS

		PAGE
1.0	SCOPE	1
2.0	APPLICABLE DOCUMENTS	2
3.0	PERFORMANCE REQUIREMENTS	3
3.1	General Description	3
3.1.1	Instruction Formats and Types	3
3.1.1.1	Formats	3
3.1.1.1.1	(N/A)	3
3.1.1.1.2	(N/A)	3
3.1.1.1.3	(N/A)	3
3.1.1.1.4	Format 4	3
3.1.1.1.5	Format 5	4
3.1.1.1.6	Format 6	4
3.1.1.1.7	Format 7	4
3.1.1.1.8	(N/A)	4
3.1.1.1.9	Format 9	4
3.1.1.1.10	Format A	4
3.1.1.1.11	Format B	5
3.1.1.1.12	Format C	5
3.1.1.1.13	Format D	5
3.1.1.1.14	Format E	6
3.1.1.1.15	Format F	6
3.1.1.2	Instruction Types	7
3.1.1.2.1	Register Instructions (RG)	7
3.1.1.2.2	Index Instructions (IN)	7
3.1.1.2.3	Branch Instructions (BR)	7
3.1.1.2.4	Stream Instructions (SM)	8
3.1.1.2.5	(N/A)	8
3.1.1.2.6	(N/A)	8
3.1.1.2.7	(N/A)	8
3.1.1.2.8	(N/A)	8
3.1.1.2.9	Monitor Instructions (MN)	9
3.1.1.2.10	Non-Typical Instruction (NT)	9
3.1.2	Addressing	9
3.1.2.1	Memory Hierarchy Addressing	12
3.1.2.1.1	Register File Addressing	12

(continued)

TABLE OF CONTENTS

PAGE

3.1.2.1.2 Main Memory Addressing 12
3.1.2.1.3 Intermediate Memory Addressing 13
3.1.2.1.4 Backing Store Addressing 13
3.1.2.1.5 Illegal Addresses. 13

3.1.2.2 Instruction Addressing 14

3.1.3 Termination Rules. 14

3.1.3.1 Stream Instruction Termination 14
3.1.3.2 (N/A). 15
3.1.3.3 (N/A). 15
3.1.3.4 (N/A). 15

3.1.4 Definitions and Rules. 15

3.1.4.1 Overlap of Operand and Result Fields 15
3.1.4.2 Self-Modifying Programs, Undefined Instructions and Undefined Operands 15

3.1.4.2.1 Self-Modifying Programs. 15
3.1.4.2.2 Illegal Instructions 16
3.1.4.2.3 Undefined Instructions 16
3.1.4.2.4 (N/A). 16
3.1.4.2.5 No op Instructions 16

3.1.4.3 Floating-Point Format. 16

3.1.4.3.1 32-Bit Floating-Point Format 17
3.1.4.3.2 64-bit Floating-Point Format 19

3.1.4.4 End Cases. 20
3.1.4.5 Floating-Point Compare Rules 21

3.1.4.5.1 One or Both Operands Indefinite. 21
3.1.4.5.2 Neither Operand Indefinite but One or Both Operands Machine Zero. 21
3.1.4.5.3 Neither Operand Indefinite Nor Machine Zero. . . 22
3.1.4.6 Upper and Lower Results. 24

3.1.4.6.1 Right Normalization. 24
3.1.4.6.2 Floating-Point Add 25
3.1.4.6.3 Floating-Point Subtract. 26
3.1.4.6.4 Results of the Floating-Point Multiply Instruction. 30
3.1.4.6.5 Results of the Floating-Point Divide Instruction 31
3.1.4.6.6 Normalized Upper Results 31

(continued)

TABLE OF CONTENTS

PAGE

3.1.4.6.7 (N/A). 32

3.1.4.7 (N/A). 32
3.1.4.8 (N/A). 32
3.1.4.9 (N/A). 32
3.1.4.10 (N/A). 32
3.1.4.11 Operand Size Definitions 32

3.1.5 Item Count (field lengths, indices, etc.). 33

3.1.6 Data Flag Branch Register. 34

3.1.6.1 General Description. 34
3.1.6.2 Register Description 35

3.1.6.2.1 Free Flags 36
3.1.6.2.2 Data Flags 36
3.1.6.2.3 Mask Bits. 36
3.1.6.2.4 Product Bits 37
3.1.6.2.5 Data Flag Branch Enable Bit. 38
3.1.6.2.6 Conditional Inhibits 38
3.1.6.2.7 Data Flag Register Word 0 Bit Assignments. 39
3.1.6.2.8 Data Flag Register Word 1 Bit Assignments. 44
3.1.6.2.9 Data Flag Register Word 2 Bit Assignments. 46
3.1.6.2.10 Data Flag Register Word 3 Bit Assignments. 48
3.1.6.2.11 Data Flag Usage by Instruction 50

3.1.6.3 Data Flag Branch (DFB) 55

3.1.7 Register File. 56
3.1.8 Real-Time Counters 63

3.1.8.1 Free-Running Clock 63
3.1.8.2 Monitor Interval Timer 64
3.1.8.3 Job Interval Timer 64

3.1.9 (N/A). 65
3.1.10 Exchange Operations and Invisible Package. 65

3.2 Performance Characteristics. 67

3.2.1 Instruction Descriptions 67

3.2.1.1 00 4 NA MN IDLE. 67
3.2.1.2 01 ILLEGAL. 68
3.2.1.3 02 ILLEGAL. 68
3.2.1.4 03 ILLEGAL. 68

(continued)

TABLE OF CONTENTS

						PAGE
3.2.1.5	04	4	64	NT	BREAKPOINT-MAINTENANCE.	68
3.2.1.6	05	ILLEGAL.				70
3.2.1.7	06	7	NA	MN	FAULT TEST - MAINTENANCE.	70
3.2.1.8	07	ILLEGAL.				71
3.2.1.9	08	4	NA	MN	INPUT/OUTPUT PER R.	72
3.2.1.10	09	4	64	BR	EXIT FORCE.	72
3.2.1.11	0A	4	64	MN	TRANSMIT (R) TO MONITOR INTERVAL TIMER	72
3.2.1.12	0B	ILLEGAL.				72
3.2.1.13	0C	ILLEGAL.				72
3.2.1.14	0D	ILLEGAL.				72
3.2.1.15	0E	4	64	MN	TRANSLATE EXTERNAL INTERRUPT. . .	73
3.2.1.16	0F	ILLEGAL.				74
3.2.1.17	10	A	64	RG	CONVERT BCD TO BINARY, FIXED LENGTH.	74
3.2.1.18	11	A	64	RG	CONVERT BINARY TO BCD, FIXED LENGTH.	74
3.2.1.19	12	7	8	NT	LOAD BYTE; (T) PER (S), (R) . . .	74
3.2.1.20	13	7	8	NT	STORE BYTE; (T) PER (S), (R). . .	74
3.2.1.21	14	ILLEGAL.				75
3.2.1.22	15	ILLEGAL.				75
3.2.1.23	16	ILLEGAL.				75
3.2.1.24	17	ILLEGAL.				75
3.2.1.25	18	ILLEGAL.				75
3.2.1.26	19	ILLEGAL.				75
3.2.1.27	1A	ILLEGAL.				75
3.2.1.28	1B	ILLEGAL.				75
3.2.1.29	1C	ILLEGAL.				75
3.2.1.30	1D	ILLEGAL.				75
3.2.1.31	1E	ILLEGAL.				75
3.2.1.32	1F	ILLEGAL.				75
3.2.1.33	20	7	64	RG	SHIFT (R) AND (R+1) PER S TO (T) AND (T+1)	76
3.2.1.34	21	7	64	RG	SHIFT (R) AND (R+1) PER (S) TO (T) AND (T+1)	76
3.2.1.35	22	ILLEGAL.				77
3.2.1.36	23	ILLEGAL.				77
3.2.1.37	24	7	32	NT	INTERMEDIATE MEMORY LOAD; (T) PER (S), (R).	77
3.2.1.38	25	7	32	NT	INTERMEDIATE MEMORY STORE; (T) PER (S), (R).	77
3.2.1.39	26	7	64	NT	INTERMEDIATE MEMORY LOAD; (T) PER (S), (R).	78
3.2.1.40	27	7	64	NT	INTERMEDIATE MEMORY STORE; (T) PER (S), (R).	78
3.2.1.41	28	ILLEGAL.				78
3.2.1.42	29	ILLEGAL.				78

(continued)

TABLE OF CONTENTS

						PAGE
3.2.1.43	2A	ILLEGAL				78
3.2.1.44	2B	4	64	RG	ADD TO LENGTH FIELD	78
3.2.1.45	2C	4	64	RG	LOGICAL EXCLUSIVE OR (R), (S), TO (T)	78
3.2.1.46	2D	4	64	RG	LOGICAL AND (R), (S), TO (T)	78
3.2.1.47	2E	4	64	RG	LOGICAL INCLUSIVE OR (R), (S), TO (T)	78
3.2.1.48	2F	9	1	BR	REGISTER BIT BRANCH AND ALTER	79
3.2.1.49	30	7	64	RG	SHIFT (R) PER S TO (T)	80
3.2.1.50	31	7	64	BR	INCREASE(R) AND BRANCH IF(R) <> 0.	80
3.2.1.51	32	9	1	BR	BIT BRANCH AND ALTER	81
3.2.1.52	33	B	1	BR	DATA FLAG REGISTER BIT BRANCH AND ALTER	82
3.2.1.53	34	4	64	RG	SHIFT(R) PER (S) TO (T)	83
3.2.1.54	35	7	64	BR	DECREASE (R) AND BRANCH IF (R) <> 0.	84
3.2.1.55	36	7	64	BR	BRANCH AND SET (R) TO NEXT INSTRUCTION	84
3.2.1.56	37	A	64	NT	TRANSMIT JOB INTERVAL TIMER TO (T)	84
3.2.1.57	38	A	64	IN	TRANSMIT (R BITS 00-15) TO (T BITS 00-15)	85
3.2.1.58	39	A	64	NT	TRANSMIT REAL-TIME CLOCK TO (T)	85
3.2.1.59	3A	A	64	NT	TRANSMIT (R) TO JOB INTERVAL TIMER	85
3.2.1.60	3B	7	64	BR	DATA FLAG REGISTER LOAD/STORE	85
3.2.1.61	3C	4	32	NT	HALF-WORD INDEX MULTIPLY (R)*(S) TO (T)	86
3.2.1.62	3D	4	64	NT	INDEX MULTIPLY (R)*(S) TO (T)	86
3.2.1.63	3E	6	64	IN	ENTER (R) WITH I (16 BITS)	86
3.2.1.64	3F	6	64	IN	INCREASE (R) BY I (16 BITS)	86
3.2.1.65	40	4	32	RG	ADD U; (R)+(S) TO (T)	87
3.2.1.66	41	4	32	RG	ADD L; (R)+(S) TO (T)	87
3.2.1.67	42	4	32	RG	ADD N; (R)+(S) TO (T)	87
3.2.1.68	43	ILLEGAL				87
3.2.1.69	44	4	32	RG	SUB U; (R)-(S) TO (T)	87
3.2.1.70	45	4	32	RG	SUB L; (R)-(S) TO (T)	87
3.2.1.71	46	4	32	RG	SUB N; (R)-(S) TO (T)	87
3.2.1.72	47	ILLEGAL				87
3.2.1.73	48	4	32	RG	MPY U; (R)*(S) TO (T)	87
3.2.1.74	49	4	32	RG	MPY L; (R)*(S) TO (T)	87

(continued)

TABLE OF CONTENTS

						PAGE
3.2.1.75	4A	ILLEGAL				87
3.2.1.76	4B	4	32	RG	MPY S; (R)*(S) TO (T)	87
3.2.1.77	4C	4	32	RG	DIV U; (R)/(S) TO (T)	87
3.2.1.78	4D	6	32	IN	HALF-WORD ENTER R WITH I(16 BITS)	87
3.2.1.79	4E	6	32	IN	HALF-WORD INCREASE R BY I(16 BITS)	88
3.2.1.80	4F	4	32	RG	DIV S; (R)/(S) TO (T)	88
3.2.1.81	50	A	32	RG	TRUNCATE; (R) TO (T)	88
3.2.1.82	51	A	32	RG	FLOOR; (R) TO (T)	89
3.2.1.83	52	A	32	RG	CEILING; (R) TO (T)	89
3.2.1.84	53	A	32	RG	SIGNIFICANT SQUARE ROOT; (R) TO (T)	90
3.2.1.85	54	4	32	RG	ADJUST SIGNIFICANCE; (R) PER (S) TO (T)	90
3.2.1.86	55	4	32	RG	ADJUST EXPONENT; (R) PER (S) TO (T)	91
3.2.1.87	56	ILLEGAL				91
3.2.1.88	57	ILLEGAL				91
3.2.1.89	58	A	32	RG	TRANSMIT; (R) TO (T)	91
3.2.1.90	59	A	32	RG	ABSOLUTE; (R) TO (T)	91
3.2.1.91	5A	A	32	RG	EXPONENT OF (R) TO (T)	92
3.2.1.92	5B	4	32	RG	PACK; (R), (S) TO (T)	92
3.2.1.93	5C	A	B	RG	EXTEND; 32-BIT (R) TO 64-BIT (T)	92
3.2.1.94	5D	A	B	RG	INDEX EXTEND; 32-BIT (R) TO 64-BIT (T)	93
3.2.1.95	5E	7	32	NT	LOAD; (T) PER (S), (R)	93
3.2.1.96	5F	7	32	NT	STORE; (T) PER (S), (R)	93
3.2.1.97	60	4	64	RG	ADD U; (R)+(S) TO (T)	93
3.2.1.98	61	4	64	RG	ADD L; (R)+(S) TO (T)	93
3.2.1.99	62	4	64	RG	ADD N; (R)+(S) TO (T)	93
3.2.1.100	63	4	64	RG	ADD ADDRESS; (R)+(S) TO (T)	94
3.2.1.101	64	4	64	RG	SUB U; (R)-(S) TO (T)	94
3.2.1.102	65	4	64	RG	SUB L; (R)-(S) TO (T)	94
3.2.1.103	66	4	64	RG	SUB N; (R)-(S) TO (T)	94
3.2.1.104	67	4	64	RG	SUB ADDRESS; (R)-(S) TO (T)	94
3.2.1.105	68	4	64	RG	MPY U; (R)*(S) TO (T)	95
3.2.1.106	69	4	64	RG	MPY L; (R)*(S) TO (T)	95
3.2.1.107	6A	ILLEGAL				95
3.2.1.108	6B	4	64	RG	MPY S; (R)*(S) TO (T)	95
3.2.1.109	6C	4	64	RG	DIV U; (R)/(S) TO (T)	95
3.2.1.110	6D	4	64	RG	INSERT BITS; (R) TO (T) PER (S)	96
3.2.1.111	6E	4	64	RG	EXTRACT BITS; (R) TO (T) PER (S)	97
3.2.1.112	6F	4	64	RG	DIV S; (R)/(S) TO (T)	98

(continued)

TABLE OF CONTENTS

						PAGE
3.2.1.113	70	A	64	RG	TRUNCATE; (R) TO (T).	98
3.2.1.114	71	A	64	RG	FLOOR; (R) TO (T)	99
3.2.1.115	72	A	64	RG	CEILING; (R) TO (T)	99
3.2.1.116	73	A	64	RG	SIGNIFICANT SQUARE ROOT; (R) TO (T).	100
3.2.1.117	74	4	64	RG	ADJUST SIGNIFICANCE; (R) PER (S) TO (T).	100
3.2.1.118	75	4	64	RG	ADJUST EXPONENT; (R) PER (S) TO (T)	101
3.2.1.119	76	A	B	RG	CONTRACT; 64-BIT (R) TO 32-BIT (T).	102
3.2.1.120	77	A	B	RG	ROUNDED CONTRACT; 64-BIT (R) TO 32-BIT (T)	103
3.2.1.121	78	A	64	RG	TRANSMIT; (R) TO (T).	103
3.2.1.122	79	A	64	RG	ABSOLUTE; (R) TO (T).	103
3.2.1.123	7A	A	64	RG	EXPONENT OF (R) TO (T).	103
3.2.1.124	7B	4	64	RG	PACK; (R), (S) TO (T)	103
3.2.1.125	7C	A	64	RG	LENGTH; (R) TO (T).	104
3.2.1.126	7D	7	64	NT	SWAP; S----->T AND R----->S.	104
3.2.1.127	7E	7	64	NT	LOAD; (T) PER (S), (R).	105
3.2.1.128	7F	7	64	NT	STORE; (T) PER (S), (R)	105
3.2.1.129	80	ILLEGAL.				105
3.2.1.130	81	ILLEGAL.				105
3.2.1.131	82	ILLEGAL.				105
3.2.1.132	83	ILLEGAL.				105
3.2.1.133	84	ILLEGAL.				105
3.2.1.134	85	ILLEGAL.				105
3.2.1.135	86	ILLEGAL.				105
3.2.1.136	87	ILLEGAL.				105
3.2.1.137	88	ILLEGAL.				105
3.2.1.138	89	ILLEGAL.				105
3.2.1.139	8A	ILLEGAL.				105
3.2.1.140	8B	ILLEGAL.				105
3.2.1.141	8C	ILLEGAL.				105
3.2.1.142	8D	ILLEGAL.				105
3.2.1.143	8E	ILLEGAL.				106
3.2.1.144	8F	ILLEGAL.				106
3.2.1.145	90	ILLEGAL.				106
3.2.1.146	91	ILLEGAL.				106
3.2.1.147	92	ILLEGAL.				106
3.2.1.148	93	ILLEGAL.				106
3.2.1.149	94	ILLEGAL.				106
3.2.1.150	95	ILLEGAL.				106
3.2.1.151	96	ILLEGAL.				106
3.2.1.152	97	ILLEGAL.				106
3.2.1.153	98	ILLEGAL.				106

(continued)

TABLE OF CONTENTS

PAGE

3.2.1.154 99 ILLEGAL. 106
3.2.1.155 9A ILLEGAL. 106
3.2.1.156 9B ILLEGAL. 106
3.2.1.157 9C ILLEGAL. 106
3.2.1.158 9D D E SM MEMORY MAP. 107
3.2.1.159 9E E E SM VECTOR READ PORT SETUP. 118
3.2.1.160 9F F E SM VECTOR ARITHMETIC 120
3.2.1.161 A0 ILLEGAL. 127
3.2.1.162 A1 ILLEGAL. 127
3.2.1.163 A2 ILLEGAL. 127
3.2.1.164 A3 ILLEGAL. 127
3.2.1.165 A4 ILLEGAL. 127
3.2.1.166 A5 ILLEGAL. 127
3.2.1.167 A6 ILLEGAL. 127
3.2.1.168 A7 ILLEGAL. 127
3.2.1.169 A8 ILLEGAL. 127
3.2.1.170 A9 ILLEGAL. 127
3.2.1.171 AA ILLEGAL. 127
3.2.1.172 AB ILLEGAL. 127
3.2.1.173 AC ILLEGAL. 127
3.2.1.174 AD ILLEGAL. 127
3.2.1.175 AE ILLEGAL. 127
3.2.1.176 AF ILLEGAL. 127
3.2.1.177 B0 C E BR COMPARE, EQUAL 128
3.2.1.178 B1 C E BR COMPARE, NOT EQUAL 128
3.2.1.179 B2 C E BR COMPARE, GREATER THAN OR EQUAL . 128
3.2.1.180 B3 C E BR COMPARE, LESS THAN 128
3.2.1.181 B4 C E BR COMPARE, LESS THAN OR EQUAL. . . 128
3.2.1.182 B5 C E BR COMPARE, GREATER THAN. 128
3.2.1.183 B6 5 NA BR BRANCH TO IMMEDIATE ADDRESS;
(R) + I(48) BITS) 135
3.2.1.184 B7 ILLEGAL. 135
3.2.1.185 B8 ILLEGAL. 135
3.2.1.186 B9 ILLEGAL. 135
3.2.1.187 BA ILLEGAL. 135
3.2.1.188 BB ILLEGAL. 135
3.2.1.189 BC ILLEGAL. 135
3.2.1.190 BD ILLEGAL. 135
3.2.1.191 BE 5 64 IN ENTER (R) WITH I(48 BITS). 136
3.2.1.192 BF 5 64 IN INCREASE (R) BY I(48 BITS) 136
3.2.1.193 C0 ILLEGAL. 136
3.2.1.194 C1 ILLEGAL. 136
3.2.1.195 C2 ILLEGAL. 136
3.2.1.196 C3 ILLEGAL. 136
3.2.1.197 C4 ILLEGAL. 136
3.2.1.198 C5 ILLEGAL. 136
3.2.1.199 C6 ILLEGAL. 136

(continued)

TABLE OF CONTENTS

			PAGE
3.2.1.200	C7	ILLEGAL.	136
3.2.1.201	C8	ILLEGAL.	136
3.2.1.202	C9	ILLEGAL.	136
3.2.1.203	CA	ILLEGAL.	136
3.2.1.204	CB	ILLEGAL.	136
3.2.1.205	CC	ILLEGAL.	137
3.2.1.206	CD	5 32 IN HALF-WORD ENTER (R) WITH I(24 BITS).	137
3.2.1.207	CE	5 32 IN HALF-WORD INCREASE (R) BY I(24 BITS).	137
3.2.1.208	CF	ILLEGAL.	137
3.2.1.209	D0	ILLEGAL.	137
3.2.1.210	D1	ILLEGAL.	137
3.2.1.211	D2	ILLEGAL.	137
3.2.1.212	D3	ILLEGAL.	137
3.2.1.213	D4	ILLEGAL.	137
3.2.1.214	D5	ILLEGAL.	137
3.2.1.215	D6	ILLEGAL.	137
3.2.1.216	D7	ILLEGAL.	137
3.2.1.217	D8	ILLEGAL.	137
3.2.1.218	D9	ILLEGAL.	137
3.2.1.219	DA	ILLEGAL.	138
3.2.1.220	DB	ILLEGAL.	138
3.2.1.221	DC	ILLEGAL.	138
3.2.1.222	DD	ILLEGAL.	138
3.2.1.223	DE	ILLEGAL.	138
3.2.1.224	DF	ILLEGAL.	138
3.2.1.225	E0	ILLEGAL.	138
3.2.1.226	E1	ILLEGAL.	138
3.2.1.227	E2	ILLEGAL.	138
3.2.1.228	E3	ILLEGAL.	138
3.2.1.229	E4	ILLEGAL.	138
3.2.1.230	E5	ILLEGAL.	138
3.2.1.231	E6	ILLEGAL.	138
3.2.1.232	E7	ILLEGAL.	138
3.2.1.233	E8	ILLEGAL.	138
3.2.1.234	E9	ILLEGAL.	138
3.2.1.235	EA	ILLEGAL.	139
3.2.1.236	EB	ILLEGAL.	139
3.2.1.237	EC	ILLEGAL.	139
3.2.1.238	ED	ILLEGAL.	139
3.2.1.239	EE	ILLEGAL.	139
3.2.1.240	EF	ILLEGAL.	139
3.2.1.241	F0	ILLEGAL.	139
3.2.1.242	F1	ILLEGAL.	139
3.2.1.243	F2	ILLEGAL.	139
3.2.1.244	F3	ILLEGAL.	139

(continued)

TABLE OF CONTENTS

		PAGE
3.2.1.245	F4 ILLEGAL.	139
3.2.1.246	F5 ILLEGAL.	139
3.2.1.247	F6 ILLEGAL.	139
3.2.1.248	F7 ILLEGAL.	139
3.2.1.249	F8 ILLEGAL.	139
3.2.1.250	F9 ILLEGAL.	139
3.2.1.251	FA ILLEGAL.	139
3.2.1.252	FB ILLEGAL.	139
3.2.1.253	FC ILLEGAL.	140
3.2.1.254	FD ILLEGAL.	140
3.2.1.255	FE ILLEGAL.	140
3.2.1.256	FF ILLEGAL.	140
3.2.2	Instruction Execution Times.	140
4.0	TEST REQUIREMENTS (not applicable)	141
5.0	PREPARATION FOR DELIVERY (not applicable). . . .	142
6.0	NOTES.	143
6.1	ASCII/EBCDIC Reference Charts.	143
APPENDIX A		148
A1.0	SCOPE.	148
A2.0	SELF-MODIFYING PROGRAMS.	148
A3.0	INSTRUCTION STACK.	149
A4.0	(N/A).	149
A5.0	VECTOR FORMATS	149
A6.0	DATA FLAG BRANCH	150

(continued)

1.0 SCOPE

This specification for the CDC FLOW MODEL PROCESSOR (FMP) is to be used in conjunction with the CDC STAR-100 Computer Specifications. It is assumed that the reader is familiar with the concepts and terminology described in those documents.

This is NOT a reference manual for user's groups. This document is written expressly for logic designers and diagnostic programmers.

2.0 APPLICABLE DOCUMENTS

10354637 CDC FLOW MODEL PROCESSOR Functional Computer Specification

3.0 PERFORMANCE REQUIREMENTS

3.1 General Description

3.1.1 Instruction Formats and Types

3.1.1.1 Instruction Formats - all fields are 8 bits unless otherwise specified.

3.1.1.1.1 (N/A)

3.1.1.1.2 (N/A)

3.1.1.1.3 (N/A)

3.1.1.1.4 Format 4

F Function	R Source 1	S Source 2	T Desti- nation

3.1.1.1.5 Format 5

F Function	R Destination	I48

3.1.1.1.6 Format 6

F Function	R Destination	I16

3.1.1.1.7 Format 7

F Function	R **	S **	T **

**Described where used

3.1.1.1.8 (N/A)

3.1.1.1.9 Format 9

F Function	G Sub-Function	S **	T **

3.1.1.1.10 Format A

F Function	R Register	*	T Register

3.1.1.1.11 Format B

Fields N and I are defined in the instruction descriptions where the format is used.

F Function	G Sub-Function	N 2	I 6	T Base Address

3.1.1.1.12 Format C

```
         0   4567
--------------------------------------------------------------------------------
| F      | |||||| X      | A      | Y   | B      | Z      | C      |
|Function|*||||||Register|Register|Index| Base   |Register|Register|
|        | ||||||        |        |     |Address |        |        |
--------------------------------------------------------------------------------
```

* Unused area must be cleared to zeros.
** Described where used

3.1.1.1.13 Format D

All fields are defined in the instruction descriptions where the format is used.

```
 0      0 0 1 1   1 1  1 1   2 2 2 2  2 3 3
 0      7 8 0 1   5 6  8 9   3 4 5 7  9 0 1
+--------+---------+----------+-----------+
|        |   |     | |  |     | |   |  | | |
|   F    | S |  K  |X| A|  Z  |B| Y | C|D|E|
|        |   |     | |  |     | |   |  | | |
+--------+---------+----------+-----------+

                       3       3 4     4 4      5 5       6
                       2       9 0     7 8      5 6       3
                       ------------------------------------
                       |       |       |        |        |
                       |   T   |   U   |   V    |   W    |
                       |       |       |        |        |
                       ------------------------------------
```

3.1.1.1.14 Format E

All fields are defined in the instruction descriptions where the format is used.

```
 0      0 0 1 1   1 1 1  1 2 2  2 2 2  2 2 2  3
 0      7 8 0 1   5 6 7  9 0 1  3 4 5  7 8 9  1
+--------+---------+---------+-------------+
|        | |  |    |  |  |   | | |   | | |   | |  |
|   F    | X  |   K |L |A | E |M|B| G |N|C| H |P|D| J|
|        | |  |    |  |  |   | | |   | | |   | |  |
+--------+---------+---------+-------------+

                          3      3 4      4 4      5 5      6
                          2      9 0      7 8      5 6      3
                         +--------+--------+--------+--------+
                         |        |        |        |        |
                         |   T    |   U    |   V    |   W    |
                         |        |        |        |        |
                         +--------+--------+--------+--------+
```

3.1.1.1.15 Format F

All fields are defined in the instruction descriptions where the format is used.

```
 0      0 0  1 1   1  1  1     2  2  2  2  3
 0      7 8  0 1   5  7  8     3  5  7  9  1
+--------+----------+-----------+-----------+
|        | | |      |   |       |  |  |  |  |
|   F    |E| G|  Z  | H |   S   | A| B| C| D|
|        | | |      |   |       |  |  |  |  |
+--------+----------+-----------+-----------+

                3 3 3 3 3  3 4 4 4 4 4  4 4      5 5      6
                2 3 4 5 6  9 0 1 2 3 4  7 8      5 6      3
               +-----------+-----------+--------+--------+
               | | | | |   | | | | |   |        |        |
               |T|L|N|X| Q |U|M|P|Y| R |   V    |   W    |
               | | | | |   | | | | |   |        |        |
               +-----------+-----------+--------+--------+
```

3.1.1.2 Instruction Types

3.1.1.2.1 Register Instructions (RG)

In the register instructions, all operand sources and all result destinations are registers. R, S, and T each designate one of 256 registers.

A register may be used to hold one or both source operands as well as the result. Special case: if register 00 is designated as a source or result register, see Section 3.1.7.

Unless stated differently in the instruction description, in all register-to-register operations the contents of the source registers are unchanged and the destination register is cleared before the result is transferred into it.

3.1.1.2.2 Index Instructions (IN)

The index instructions are used primarily to perform numerical calculations on field lengths or addresses.

The term, replace, means replace only the specified bits. The phrase, replace the right-most 48 bits ..., implies that the left-most 16 bits are not altered.

3.1.1.2.3 Branch Instructions (BR)

Branch conditions may be determined by examining single bits, a 48-bit index, 32-bit floating-point operands, or 64-bit floating-point operands. A special branch is provided to enter and leave the monitor program. All item counts in branch instructions are in half-words.

3.1.1.2.4 Stream Instructions (SM)

Stream instructions operate on ordered sets of data, executing in one or more of the streaming units: Main Map, Intermediate Map, Vector Streaming, and Vector Units. In general, the stream instructions perform one of two functions: mapping (reordering) data in memory or arithmetic operations on ordered sets of data (vectors). For both functions, data is taken from a memory and results are returned to a memory.

Mapping operations are executed in the Main Map Unit (for mapping Main Memory), in the Intermediate Memory, or in both Map Units operating together (for some mapping functions which can be done from/to Main Memory to/from Intermediate Memory).

The Vector Units and Vector Streaming Unit always operate together for execution of the arithmetic operations. The Vector Streaming Unit serves as the control element to supply operand streams from Main Memory to the vector pipelines and to return result streams to Main Memory.

Stream instructions are 64 bits in length, broken into fields, the number and size of which depend on the actual instruction. The fields are defined in each stream instruction description for that particular instruction. In general, however, they include a function code, F, a suboperation code, S, and register designators for specifying source of setup information from the Register File; these designators are T, U, V, and W, some of which may not be used for a particular instruction. See the individual stream instruction descriptions for more detailed information on these and other fields.

3.1.1.2.5 (N/A)

3.1.1.2.6 (N/A)

3.1.1.2.7 (N/A)

3.1.1.2.8 (N/A)

3.1.1.2.9 Monitor Instructions (MN)

Monitor instructions perform as described only when in monitor mode. When not in monitor mode, the monitor instructions perform as an illegal instruction would (see Section 3.1.4.2.2).

3.1.1.2.10 Non-Typical Instruction (NT)

The format and operation of these instructions are completely described under the individual instruction descriptions.

3.1.2 Addressing

Groups of bits in an address should be thought of as addressing various units of storage as illustrated in the chart below.

```
                 16----------->  54 55 56 57 58 59 60 61 62 63
                |---------\  \---------------------------------|
Bit position    |          \  \    |  |  |  |  |  |  |  |  |    |
in a register   |-----------\  \-------------------------------|
or an in-       |                  |        |  |        |       |
struction       |                  |        |  |        |       |
word            |    Address of    |        |  |        |       |
                |<-----Sword----->|        |  |        |       |
                |                           |  |        |       |
                |<--Address of Word------->|  |        |       |
                |                              |        |       |
                |<--Address of Half-Word---->|        |       |
                |                                       |       |
                |<--------Address of Byte---------->|       |
                |                                               |
                |<--------Address of Bit-------------------->|
                |                                               |
```

Within a word, bits, bytes, and half-words are always numbered from left to right. The lowest addressed bit, byte, or half-word is always the left-most bit, byte, or half-word in the word.

(continued)

3.1.2 (Cont.)

All addresses are 48-bit quantities and contain enough information to reference a specific bit. Depending on the usage of an address, a certain number of the right-most bits in the address are ignored. For example, if a byte is being read, the right-most three bits of the address being used to reference it are ignored. Depending on the instruction, operands are counted on a bit, byte, half-word, or word basis.

```
    |<------half-word 0------------>|<------half-word 1-------->|
    |                               |                           |
    -------------------------------------------------------------
    |      |       |        |       |      |      |      |      |
    |byte 0| byte 1| byte 2 |byte 3 |byte 4|byte 5|byte 6|byte 7|
    -------------------------------------------------------------
bit 0     7 8    15 16    23 24    31 32  39 40  47 48  55 56  63
```

The above figure illustrates the relative location of each bit, byte, and half-word within a 64-bit word.

If it is necessary to add an address and an index, the index is shifted left end off until it is properly aligned with the address. Binary zeros are attached to the right end of the quantity being shifted.

The result of the addition always addresses a quantity having the same unit as the index. For instance, if a byte count is added to any address, the result references a byte. This means that the right-most three bits of the address will be ignored. The following chart summarizes the process of adding an index to an address and shows which bits are ignored in the resulting address.

(continued)

3.1.2 (Cont.)

```
                                          16                    57 58 59 60 61 62 63
                                          ----------------------------------------
   Base Address--------------->|                     |  |  |  |  |  |  |
                                          ----------------------------------------
                                                                 |  |     |        |
       /                                                         |  |     |        |
       |              16            22                     63|  |     |        |
       |              ----------------------------------    |     |        |
       |A. words      | **          |words                | 0|  0 0 |  0 0 0 |
       |              ----------------------------------    |     |        |
       |                   16       21                        63|     |        |
       |                   ---------------------------------    |        |
Index -|B. half-           | **     |half-words                 | 0 0 |  0 0 0 |
      /    words           ---------------------------------    |        |
      \                          16  19                             63|        |
       |                         ----------------------------------        |
       |C. bytes                 |**|bytes                          | 0 0 0 |
       |                         ----------------------------------        |
       |                            16                                       63|
       |                            ----------------------------------------
       |D. bits                     |bits                                      |
       |                            ----------------------------------------
       \

                                    16                    57 58 59 60 61 62 63
       /                            ----------------------------------------
       |                            |                     |  |  |  |  |  |  |
       |                            ----------------------------------------
       |                            |                     |  |     |        |
       |A. words                    |<---Bits used->|<---  *  --------->|
       |                            |                     |  |     |        |
Result-|                            |                        |     |        |
ant    |B. half-                    |<---Bits used---->|<------- * ->|
address-|    words                  |                        |     |        |
es     |                            |                              |        |
       |C. bytes                    |<---Bits used---------->|<- * ->|
       |                            |                              |        |
       |                            |                                       |
       |D. bits                     |<--------Bits used------------->|
       \                            |                                       |
```

* These bits in the resultant address are ignored.

** These bits in the index are shifted off and do not enter the address calculation.

3.1.2.1 Memory Hierarchy Addressing

There are four levels of memory accessible to the programmer: Register File, Main Memory, Intermediate Memory, and Backing Store. These memories can be addressed by instructions in different ways, dependent on the memory level.

The actual amount of physical memory present is determined by the specific machine configuration. Memory not actually in existence causes an operand abort to occur at the time of reference.

3.1.2.1.1 Register File Addressing

Register File addresses are generally contained within the instruction itself; such direct addresses are called register designators, each designator being assigned a name such as R, S, or T. A register designator in an instruction is an 8-bit field and therefore can address any one of 256 registers (64-bit or 32-bit).

The only exception to a register file address being contained within the instruction is the SWAP (7D) instruction. In this case, one register designator is contained in the right-most eight bits (bits 56 to 63) of a register.

3.1.2.1.2 Main Memory Addressing

Main memory addresses are contained in 64-bit registers in the Register File. Thus an instruction can reference memory indirectly by giving the appropriate register file address, which points to the register containing the desired memory address.

An address field of 27 bits is established permitting access to 134,217,728 64-bit words of Main Memory. Since all addresses are bit addresses, the right-most 33 bits (bits 31-63) of the register are used.

3.1.2.1.3 Intermediate Memory Addressing

Intermediate Memory is addressed in the same manner as Main Memory with the addresses contained in, and supplied from, 64-bit registers in the Register File. A larger address field of 29 bits allows access to 536,870,912 64-bit words of Intermediate Memory. All addresses are bit addresses, thus the right-most 35 bits (bits 29 to 63) of the register are used.

3.1.2.1.4 Backing Store Addressing

The Backing Store is essentially like an I/O device and transfers data to/from Intermediate Memory in blocks of 32,768 words (see Functional Specification 10354637 for more detailed information). While it is accessible under program control, it is not addressed specifically by instruction. Instead, addresses are stored in some location in Intermediate Memory and the backing store control (Swap Unit) is notified of the location.

The address provided is a block address (32,768 words) and the field is 16 bits allowing access to 65,536 blocks, or 2,147,483,648 64-bit words. Since all addresses are bit-addresses, the right-most 37 bits (bits 27 to 63) are used.

3.1.2.1.5 Illegal Addresses

Main memory bit addresses 0 through 400000_{16} are reserved for the operating system. Any reference to this address range by a job mode program results in a job mode illegal abort of the program in execution.

Main memory bit addresses 0 through 4000_{16} are reserved for the storage of the monitor's register file. Any reference to this area by a monitor mode memory access will cause a monitor mode illegal abort.

When addressing non-existent areas of memory, the FMP will generate an operand abort.

3.1.2.2 Instruction Addressing

Instructions are addressed on full-word and half-word boundaries. The instruction address counter will, therefore, be incremented by a half-word after executing a 32-bit instruction and by a full word after executing a 64-bit instruction. This allows instructions to be packed contiguously in storage. The following chart illustrates the various ways instructions may be packed within 64-bit words.

bit position

0 … 31	32 … 63
32-bit inst.	64-bit inst. upper
64-bit inst. lower	64-bit inst. upper
64-bit inst. lower	32-bit inst.
64-bit instruction	
32-bit inst.	32-bit inst.

Note that a branch is possible to any of the instructions. The lower 5 bits in any branch address will always be interpreted as zeros.

3.1.3 Termination Rules

For instructions which terminate upon exhausting the length of a data field, data string, or vector: if that item is exhausted prior to the first operand fetch, the instruction becomes a No op; no data is fetched and no data flags are altered.

3.1.3.1 Stream Instruction Termination

Most stream instructions terminate when the result vector is exhausted. Source vectors which are exhausted before the result vector is exhausted are extended or repeated, as required, designated by setup data in the instruction. Those vector operations designated as recursive (Sum, Dot Product, etc.) terminate on the input length because only one result is returned from each pipeline, regardless of input length.

3.1.3.2 (N/A)

3.1.3.3 (N/A)

3.1.3.4 (N/A)

3.1.4 Definitions and Rules

3.1.4.1 Overlap of Operand and Result Fields

If the result field overlaps a source field such that elements of the result are stored in the source field before elements in this portion of the source field are read, undefined results may occur. That is, the source elements may be the original elements or they may be the newly-stored elements. The instruction's results may become undefined. Note that some specific instructions prohibit any overlap of source and destination fields. This restriction is included in the appropriate instruction descriptions.

3.1.4.2 Self-Modifying Programs, Undefined Instructions and Undefined Operands

3.1.4.2.1 Self-Modifying Programs [A2.0]

As a general rule, self-modifying programs are not allowed. See Appendix A2.0 for further details.

3.1.4.2.2 Illegal Instructions

An instruction with an unused function code is termed an illegal instruction and causes the following:

A. If in monitor mode, an automatic branch to the address specified by the contents of absolute register 4 is executed.

B. If in job mode, an exchange to monitor mode is performed with execution beginning at the address specified by the contents of absolute register 3.

3.1.4.2.3 Undefined Instructions

The instructions with a defined F code but which either have undefined bits set or specify an undefined operation cause undefined results.

3.1.4.2.4 (N/A)

3.1.4.2.5 No op Instructions

The instructions that are defined as No op (no operation) instructions do not fetch data and do not alter data flags.

3.1.4.3 Floating-Point Format

3.1.4.3.1 32-Bit Floating-Point Format

```
      -- bit 0, exponent sign bit
      |
      V
     _____________________
     | |                 |
     | |                 |
     --------------------- * exponent binary point
     |   1             7 |
     |                   ||-- bit 8, coefficient sign bit
     |                   || 
     |                   |V
     |                   |-------------------------------|
     |                   |   |                           |
     |                   |   |                           |
     |                   |-------------------------------|* coef.
     |                   |   9                        31 | binary
     |                   |                               | point
     |                   |                               |
     -----------------------------------------------------
     | 8-bit signed      |      24-bit signed            |
     |  exponent         |       coefficient             |
     -----------------------------------------------------
      0              7  8                             31
bit  \_________________________________________________/
                              V
                 32-bit floating-point number
```

There are two 32-bit half-words in every 64-bit word. A 32-bit floating-point number occupies a half-word.

A zero is a positive sign bit and a one is a negative sign bit for both the exponent and the coefficient.

Both the exponent and the coefficient are expressed as two's complement signed integers. Numbers are of the form $(c)*2^{x}$ where c is the 24-bit signed coefficient, x is the 8-bit signed exponent, and the base is 2.

(continued)

3.1.4.3.1 (Cont.)

The range of useful coefficients is from 800000_{16} to $7FFFFF_{16}$.

This represents numbers of the range $-(2^{23})$ through $+(2^{23}-1)$.

The range of useful exponents is from 90_{16} to $6F_{16}$ which is from minus 112_{10} to plus 111_{10}. The values of 70_{16} through $8F_{16}$ all fall into a special end case range as defined by the following table. X is any hexadecimal digit.

Element	Representation
Machine Zero	$8XXXXXXX_{16}$
Indefinite	$7XXXXXXX_{16}$

Examples of 32-bit floating-point format represented in base 16.

+1	00	000001
+1 normalized	EA	400000
-1	00	FFFFFF
-1 normalized	E9	800000
$+256_{10}$	00	000100

A floating-point number is normalized if the coefficient sign bit is different from the next bit to the right. This condition implies that the coefficient has been shifted to the left as far as possible. Note that an all zero coefficient requires special attention for normalized operations.

3.1.4.3.2 64-bit Floating-Point Format

```
      --- bit 0, exponent sign bit
      |
      V
     ____________________
     | |                |
                          * exponent binary point
     |   1          15 |
                         |-- bit 16, coefficient sign bit
     |                 | |
                         V
     |                 |------------------------------|
                       |   |                          |
     |                 |   |                          |
                       |------------------------------|* coef.
     |                 |   17                      63 | binary
                                                        point
     |                 |                              |
     --------------------------------------------------
     | 16-bit signed   |        48-bit signed         |
     |   exponent      |         coefficient          |
     --------------------------------------------------
      0            15  16                          63
bit  \_________________________ ____________________/
                               V
                   64-bit floating-point number
```

A 64-bit floating-point number is contained in a 64-bit word.

A zero is a positive sign bit and a one is a negative sign bit for both the exponent and the coefficient.

Both the exponent and the coefficient are expressed as two's complement signed integers. Numbers are of the form $(c)*2^{x}$ where c is the 48-bit signed coefficient, x is the 16-bit signed exponent, and the base is 2.

The range of useful coefficients is from $8000\ 0000\ 0000_{16}$ to $7FFF\ FFFF\ FFFF_{16}$. This represents numbers of the range $-(2^{47})$ through $+(2^{47}-1)$.

(continued)

3.1.4.3.2 (Cont.)

The range of useful exponents is from 9000_{16} to $6FFF_{16}$ which is from minus $28,672_{10}$ to plus $28,671_{10}$. The values of 7000_{16} through $8FFF_{16}$ all fall into a special end case range as defined by the following table. X is any hexadecimal digit.

Element	Representation
Machine Zero	$8XXXXXXXXXXXXXXX_{16}$
Indefinite	$7XXXXXXXXXXXXXXX_{16}$

Examples of 64-bit floating-point format represented in base 16.

+1	0000 0000 0000 0001
+1 normalized	FFD2 4000 0000 0000
-1	0000 FFFF FFFF FFFF
-1 normalized	FFD1 8000 0000 0000
$+256_{10}$	0000 0000 0000 0100

A floating-point number is normalized if the coefficient sign bit is different from the next bit to the right. This condition implies that the coefficient has been shifted to the left as far as possible. Note that an all zero coefficient requires special attention for normalized operations.

3.1.4.4 End Cases

If indefinite is used as an operand in a floating-point instruction, both the upper and the lower results are indefinite.

For the cases listed below, 0 represents machine zero and N represents an operand which is neither machine zero nor indefinite.

$0 \pm 0 = 0$	$0 * 0 = 0$	$0 / 0 =$ Indefinite
$0 \pm N = \pm N$	$0 * N = 0$	$0 / N = 0$
$N \pm 0 = N$	$N * 0 = 0$	$N / 0 =$ Indefinite

3.1.4.5 Floating-Point Compare Rules

Several of the instructions compare two floating-point operands for:

a.	equality	$(r) = (s)$
b.	non-equality	$(r) <> (s)$
c.	greater than or equal to	$(r) \geq (s)$
d.	less than	$(r) < (s)$

For these examples, the first operand is represented by (r) and the second operand by (s).

3.1.4.5.1 One or Both Operands Indefinite

If one operand is indefinite, no compare condition is met since indefinite is not: greater than, less than, equal to, nor not equal to any other operand.

If both operands are indefinite, the $(r) = (s)$ and the $(r) \geq (s)$ conditions are met since indefinite is defined equal to indefinite.

3.1.4.5.2 Neither Operand Indefinite but One or Both Operands Machine Zero

Any non-indefinite, non-machine zero operand with a positive, non-zero, coefficient is strictly greater than machine zero.

Any non-indefinite, non-machine zero operand with a negative coefficient is strictly less than machine zero.

Machine zero is equal only to itself and any number having a finite exponent and an all zero coefficient.

3.1.4.5.3 Neither Operand Indefinite Nor Machine Zero

If the signs of the coefficients of the two operands are unlike, the operands are unequal and the operand with the positive coefficient is the larger of the two.

If the signs of the two coefficients are alike, a floating-point subtract upper is performed, r - s.

Condition met criteria are analyzed as follows:

a. If the upper 48 bits of the result coefficient are all zeros (r) = (s)
b. If the upper 48 bits of the result coefficient are not all zeros (r) <> (s)
c. If the result coefficient is positive (r) $\geq$ (s)
d. If the result coefficient is negative (r) < (s)

The above criteria (a and b) for equality and non-equality do not guarantee for r = s, that s = r when the following is true:

a. The operands have unequal exponents.
b. "1" bits exist in any of the right-most bit positions of the coefficient which will be shifted off the right during alignment of the smaller exponent. For example:

```
         0   16                          63
         ---------------------------------
    r =  |0004|                          |
         ---------------------------------

         ---------------------------------
    s =  |0000|                       |X|
         ---------------------------------
```

Exponent difference = 4

If X = 0 then r = s implies s = r

If X <> 0 then if r = s, s <> r
or if s = r, r <> s

(continued)

3.1.4.5.3 (Cont.)

The order of events of the floating-point subtract upper is first to complement the subtrahend, then align the coefficient associated with the smaller exponent and finally to perform a floating-point add operation. The following is an example of r = s but s <> r.

Operand r =	0100	0000	0000	1001	
s =	0104	0000	0000	0100	
Complement s	0104	FFFF	FFFF	FF00	
Align r	0104	0000	0000	0100	1
Add aligned r and complemented s	0104	0000	0000	0000	1

Since the upper 48 bits of the result coefficient are all zeros, the pair of operands are considered equal. However, if the operands are interchanged, the following happens:

Operand r =	0104	0000	0000	0100	
s =	0100	0000	0000	1001	
Complement s	0100	FFFF	FFFF	EFFF	
Align s	0104	FFFF	FFFF	FEFF	F
Add r and	0104	0000	0000	0100	
complemented,	0104	FFFF	FFFF	FEFF	F
aligned s	0104	FFFF	FFFF	FFFF	F

Since the upper 48 bits of the result coefficient are not all zeros, the pair of operands are considered unequal.

3.1.4.6 Upper and Lower Results

The floating-point add, subtract and multiply instructions generate a result coefficient twice the length of the source operands' coefficients. The left and right halves of this result are called the upper result (U) and the lower result (L), respectively.

The sign bit of the lower result's coefficient is not affected in a lower operation and remains at zero in two's complement arithmetic. The other bits of the lower coefficient receive no special treatment. Remember that a lower result is not meaningful alone, but it must be used in conjunction with its associated upper result.

Sections 3.1.4.6.1 - 3.1.4.6.4 are written for 64-bit operands. For 32-bit operands, substitute 47 for 95, 46 for 94, 23 for 47, and 22 for 46 where the latter numbers appear.

3.1.4.6.1 Right Normalization

When the result coefficient overflows its register, a right shift of one place is necessary. In this case, the entire 95-bit result is shifted right one place with sign extension and one is added to the exponent. This operation is known as right-normalization and it is done, when necessary, even if normalization is not explicitly specified by the instruction. This may cause exponent overflow; if so, the result is set to indefinite and data flag bit 42 may be set.

3.1.4.6.2 Floating-Point Add

Regardless of their signs, both operands' coefficients are extended to 94 bits in length, not including sign, by adding 47 zeros to the right of their binary points.

The exponents of the two operands are compared and the 94-bit coefficient of the operand having the smaller exponent is effectively shifted right one bit and its exponent increased by one, successively, until the two exponents are equal. The sign of the shifted coefficient is extended from the left to the right during the shift. Negative coefficients approach a minus one and positive coefficients approach zero as they are shifted.

The add is a 94-bit operation, not including sign. Right normalization takes place, if necessary. The coefficient for the U result is the left-most 47 bits and the coefficient for the L result is the right-most 47 bits of the 94-bit result.

The exponent for the U result is equal to the larger of the two operand exponents. Right-normalization will increase this value by one, if it occurred.

The exponent for the L result is 47_{10} less than the U result's exponent for all cases except three:

a. Right-normalization causes the U exponent to overflow; the U result is set to indefinite; the L exponent will be $6FD1_{16}$ (59_{16} in the 32-bit case).

b. If the U result's exponent minus 47_{10} causes exponent underflow, machine zero is stored as the L result.

c. If either or both operands were indefinite, the U and L results are indefinite.

3.1.4.6.3 Floating-Point Subtract

The floating-point subtract operation is performed by complementing the coefficient of the subtrahend and

performing a floating-point addition operation. The complementation is a 48-bit, two's complement operation and is performed before the operands are extended to 94 bits.

The hardware used for Floating Add or Subtract operations has an extra (or extended) coefficient sign bit. This means that the complementation of an 8000 coefficient is handled without the right shift of one and increase of the exponent by one as used elsewhere. This will cause a result (although not mathematically incorrect) which may differ from the result obtained when a right shift of one with increase of one is used, when the following conditions are met:

1. The operand of the pair having the large exponent (OR either of the two operands if their exponents are equal) must have a coefficient of 8000 ---

2. This operation must require this same operand to be complemented due to
   a. being the subtrahend in a subtract operation OR
   b. sign control in either a subtract or an add operation ---

3. The "other" operand must have a negative coefficient.

Example I A - B

```
A     60     F F F 0 0 0
B     64     8 0 0 0 0 0

                                                    | Instruction
                              CDC FMP               | Specification
                                                    |
                          Extra Sign Bit            |
                           |                        |
                           v                        |
Complement B  B  |-64  (1) 8 0 0 0 0 0              | 64   8 0 0 0 0 0
                 |                                  |
              B̄  ->64  (0) 8 0 0 0 0 0              | 65   4 0 0 0 0 0
                                                    |
Align operand    |-60  (1) F F F 0 0 0              | 60   F F F 0 0 0
with smaller     |                                  |
exponent         ->64  (1) F F F F 0 0              | 65   F F F F 8 0
                                                    |
Add A plus    A    64  (1) F F F F 0 0              | 65   F F F F 8 0
complement                                          |
of B         +B̄    64  (0) 8 0 0 0 0 0              | 65   4 0 0 0 0 0
             --    -------------------              | ---------------
                   64  (0) 7 F F F 0 0              | 65   3 F F F 8 0
                                                    |
                   64      7 F F F 0 0              | 65   3 F F F 8 0
```

(continued)

3.1.4.6.3 (Cont.)

Example II A - B

```
A     50     F F F 0 0 0
B     6F     8 0 0 0 0 0
                                                   Instruction
                       CDC FMP                   | Specification
                                                 |
                    Extra Sign Bit               |
                          |                      |
                          v                      |
Complement B B  |-6F  (1) 8 0 0 0 0 0            | 6F   8 0 0 0 0 0
                |                                |
              B̄ ->6F  (0) 8 0 0 0 0 0            | 70   4 0 0 0 0 0

Align operand   |-50  (1) F F F 0 0 0            | 50   F F F 0 0 0
with smaller    |                                |
exponent        ->6F  (1) F F F F F F            | 70   F F F F F F
                                                 |
Add A plus     A   6F  (1) F F F F F F           | 70   F F F F F F
complement                                       |
of B          +B̄   6F  (0) 8 0 0 0 0 0           | 70   4 0 0 0 0 0
              --   -------------------           | --------------
                   6F  (0) 7 F F F F F           | 70   3 F F F F F
```

If this operation is a Subtract Upper, the specified result is indefinite (with the appropriate data flags) while the CDC FMP result did not overflow. If this operation were a Subtract Normalized, note the following:

```
                                                 Instruction
                       CDC FMP                 | Specification
                                               |
Result of         6F  (0) 7 F F F F F          | 70   3 F F F F F
Subtract                                       |
Upper                                          |
                                               |
Normalize the     6F      7 F F F F F          | 6F   7 F F F F E
Upper Result                        ^                           ^
shifting zeros                      |                           |
in from the right                   -----------------------------
```

(continued)

3.1.4.6.3 (Cont.)

Note that the subtract operation is not always commutative. In other words it is not always true that (A-B) = -(B-A). This characteristic will be observed if the following is true of A and B:

a. The exponents of A and B are not equal.

b. "1" bits exist in any of the right most bit positions of the coefficient which will be shifted off the right during alignment of the smaller exponent.

```
Example of (A-B) <> -(B-A):

 A =  0104   6FCB   807E   89F2
 B =  0100   6FAC   3F5D   A5FA <--
                                  |
                                  |
These two 1 bits will be shifted off during
exponent alignment.

Complement B:
-B =  0100   9053   C0A2   5A06

Align B:
-B =  0104   F905   3C0A   25A0    6

A-B:
 A =  0104   6FCB   807E   89F2
-B =  0104   F905   3C0A   25A0    6
      0104   68D0   BC88   AF92    6

A-B = 0104   68D0   BC88   AF92

Align B:
 B =  0104   06FA   C3F5   DA5F    A

Complement A:
-A = 0104    9034   7F81   760E

-(B-A):
 B = 0104    06FA   C3F5   DA5F    A
-A = 0104    9034   7F81   760E
     0104    972F   4377   506D    A
-(B-A)= 0104   68D0   BC88   AF93

  This differs from A-B in the last bit
  position.
```

3.1.4.6.4 Results of the Floating-Point Multiply Instruction

When two floating-point numbers are multiplied, the lower result retains the 47 least significant product bits generated. The sign bit of the lower result is always set to zero and the exponent of the lower result is the sum of the two source operands' exponents with the exceptions listed below:

The upper result retains the 47 product bits immediately to the left of the bits retained by the lower product. The sign of the upper product's coefficient follows the normal rules of algebra. The exponent of the upper result is the sum of the two source operands' exponents plus 47 with the following exceptions:

a. The sum of the source operands' exponents (plus 47_{10}, if upper result) exceed $6FFF_{16}$ for which case the result exponent is set to indefinite.

b. The sum of the source operands' exponents (plus 47_{10}, if upper result) is less than 9000_{16} for which case the result exponent is set to machine zero.

c. Either or both operands are indefinite for which case the result exponent is set to indefinite.

d. Neither operand is indefinite but either or both operands are machine zero, for which case the result exponent is set to machine zero.

If either operand has a coefficient of 8000 0000 0000 and an exponent of X, the operand will be treated as though its coefficient were C000 0000 0000 and its exponent were X+1.

3.1.4.6.5 Results of the Floating-Point Divide Instruction

The quotient from the divide operation is the result of dividing the prenormalized, integer coefficient of the divisor into the integer coefficient of the dividend generating a 47-bit quotient (23-bit quotient for 32-bit divide). If either operand has a coefficient of 8000 0000 0000, the operand will be handled as though its coefficient were C000 0000 0000 and its exponent increased by one. When the divide hardware normalizes the divisor coefficient, the number of places shifted left is added to the exponent of the quotient as defined below.

The exponent of the result will be given by the following equation:

$$\text{Exponent of Quotient} = (\text{Exponent of Dividend}) - (\text{Exponent of Divisor}) - (46_{10} - NC)$$

where NC is the number of places shifted left to prenormalize the divisor. For the 32-bit divide operation 22_{10} is subtracted rather than 46_{10}.

The right-most bit of the quotient is neither rounded nor adjusted. The remainder is not retained. The sign of the quotient's coefficient follows the normal rules of algebra.

3.1.4.6.6 Normalized Upper Results

The normalized add and subtract instructions generate an intermediate result identical to the final result of the Add U and the Subtract U instructions. Normalization of the intermediate, 48-bit result then takes place as follows:

The 48-bit coefficient is shifted left one bit and its exponent is decreased by one, successively, until the sign bit and the bit immediately to the right of the sign bit are different. During this shift, zeros are attached to the right end of the 48-bit coefficient. If reducing the exponent by one causes exponent underflow, the result of the normalization operation is defined as machine zero.

3.1.4.6.7 (N/A)

3.1.4.7 (N/A)

3.1.4.8 (N/A)

3.1.4.9 (N/A)

3.1.4.10 (N/A)

3.1.4.11 Operand Size Definitions

The following definitions are implied throughout the specification.

Word	- A 64-bit quantity, the address of the left-most bit always being a multiple of 64.
Half-word	- A 32-bit quantity, the address of the left-most bit always being a multiple of 32.
Byte	- An 8-bit quantity, the address of the left-most bit always being a multiple of 8.
Digit	- A 4-bit binary coded decimal number or sign. One digit per byte in zoned format and two digits per byte in packed BCD format.
Sword	- 512 bits (or 8 64-bit words).

3.1.5 Item Count (field lengths, indices, etc.)

All field lengths, indices, shift counts, etc., are item counts which specify a number of bits, digits, bytes, half-words, or words.

Where an item count other than an index is contained in a 48-bit field, there shall be at least 32 consecutive and identical sign bits. Sign bits must always be extended to the left to fill the 16-bit or 48-bit field containing it.
The item count unit is specified by the instruction title line code (see arrow).

Example

```
                             |
                             V
3.2.1.67        42    4    32    RG ADD N; (R)+(S) TO (T)
```

The 32 indicates that field lengths and indices are expressed in 32-bit half-words. Any deviation from this method of specifying the units for the various item counts would be indicated in the instruction description or in the description of the instruction type. The instruction type refers to RG (register), SM (stream), etc.

An index may be either positive or negative in sign. The maximum magnitude of an index is a function of its usage. The index is shifted to the left end-off zero/three/five/six places before the addition to the base address when the unit for the index is bits/bytes/half-words/words. Digits are not used as a unit for indices.

A field length must be positive in sign and have a magnitude of less than 2^{16}; the use of a negative field length causes that length to become strictly undefined.

3.1.6 Data Flag Branch Register [A7.0]

3.1.6.1 General Description

The data flag register is designed to give the programmer an automatic branch to a special routine for certain operands, results, conditions, etc., without his having to pay the time penalty of explicitly checking these conditions in his program. If a condition which has been previously selected to cause an automatic branch occurs during an instruction, the instruction is completed, the address of the next instruction which would have been executed is stored into the address portion of register 01 and a branch is made to the address contained in register 02. The state of the data flags in the invisible package is defined only if the program was interrupted between instructions.

3.1.6.2 Register Description

The data flag register consists of four segments, or words, each 64 bits in length. The format is shown below.

```
                 PRODUCT      MASK        DATA        FREE
                 BIT          BIT         FLAG        FLAG
                 FIELD        FIELD       FIELD       FIELD
                |           |           |           |           |
                | 16 bits   | 16 bits   | 16 bits   | 16 bits   |
                |-----------|-----------|-----------|-----------|
        Word 0  |           |           |           |           |
                -------------------------------------------------
                 0        1 1         3 3         4 4         6
                 0        5 6         1 2         7 8         3


                |           |           |           |           |
                | 16 bits   | 16 bits   | 16 bits   | 16 bits   |
                |-----------|-----------|-----------|-----------|
        Word 1  |           |           |           |           |
                -------------------------------------------------
                 0        1 1         3 3         4 4         6
                 0        5 6         1 2         7 8         3


                |           |           |           |           |
                | 16 bits   | 16 bits   | 16 bits   | 16 bits   |
                |-----------|-----------|-----------|-----------|
        Word 2  |           |           |           |           |
                -------------------------------------------------
                 0        1 1         3 3         4 4         6
                 0        5 6         1 2         7 8         3


                |           |           |           |           |
                | 16 bits   | 16 bits   | 16 bits   | 16 bits   |
                |-----------|-----------|-----------|-----------|
        Word 3  |           |           |           |           |
                -------------------------------------------------
                 0        1 1         3 3         4 4         6
                 0        5 6         1 2         7 8         3
```

Paragraphs 3.1.6.2.1 through 3.1.6.2.4 describe the four fields in general terms, while 3.1.6.2.7 through 3.1.6.2.10 define the bit assignments for words 0 through 3, respectively.

3.1.6.2.1 Free Flags

Free flags (bits 48-63) may be set, cleared and tested but will not cause an automatic data flag branch. They are not associated with unique mask bits or product bits, as are the data flags. Bits 51 and 52 of all four words have a special significance which is defined below in 3.1.6.2.5; similarly, section 3.1.6.2.6 defines a special significance for bits 53 and 54 of words 2 and 3.

3.1.6.2.2 Data Flags

Each data flag (bits 32-47) has associated with it a unique mask bit and product bit which are defined below. Data flags in word 0 indicate conditions that have occurred, usually some error detected in execution of an instruction. Once set, these flags are cleared only by execution of a Data Flag Bit Branch and Alter (33) instruction, or a Data Flag Register Load/Store (3B) instruction.

The data flags in words 1 through 3 are different in that they represent, dynamically, the state of the dependency flags and interlock flags; word 1 is assigned to the interlock flags, and words 2 and 3 are assigned to the dependency flags. These 48 bits (32-47 of words 1, 2, and 3) may not be set or cleared directly by instruction as is the case with data flags of word 0; rather, each becomes set when an instruction specifying its respective key becomes active, and becomes cleared when that instruction terminates.

3.1.6.2.3 Mask Bits

A mask bit is associated with each of the data flags. The mask bits (16-31) of each word are related only to the data flags in the same word as that of the mask bit. They have the function of selecting the conditions for which the programmer wishes an automatic data flag branch to occur. Mask bits are set and cleared only by execution of a Data Flag Bit Branch and Alter (33) instruction, or a Data Flag Register Load/Store (3B) instruction.

(continued)

3.1.6.2.3 (Cont.)

It is important to note that the associated mask bit need not be set in order to set a data flag bit. The mask function is solely one of enabling a particular data flag bit to set its associated product bit.

The order in which the mask bit and its associated data flag bit are set is immaterial, as the result is the same; that is, their associated product bit is set. However, one restriction exists for the mask bits for the interlock and dependency keys (words 1, 2, and 3). The mask bit, which enables the setting of a product bit, must be set at the time its respective data flag bit clears or the automatic data flag branch will not occur. This is due to the fact that the product bit for these flags is set on the trailing edge of the data flag, i.e., when the data flag clears.

3.1.6.2.4 Product Bits

Like the mask bits, a product bit is associated with each of the data flags. Also, the product bits (00-15) of each word are related only to the data flag bits of the same word as that of the product bit. A data flag branch is performed when one (or more) product bit(s) is (are) set and the data flag branch enable bit is set (see 3.1.6.2.5 below).

Each product bit of word 0 is the dynamic logical product of its associated data flag bit and mask bit. These product bits may not be set or cleared by instruction directly; rather, the mask bit and/or the data flag bit causing the product bit to be set must be cleared.

Product bits in words 1, 2, and 3 are not a dynamic logical product as are those of word 0. These product bits, instead, are set on the trailing edge (when the bit becomes cleared) of their respective data flag bits, provided the associated mask bit is set. Product bits in these three words, i.e., those related to the interlock and dependency keys, MUST be cleared by execution of a Data Flag Bit Branch and Alter instruction, or a Data Flag Register Load/Store instruction.

3.1.6.2.5 Data Flag Branch Enable Bit

The data flag branch enable bit, bit 52 of word 0, must be set for an automatic data flag branch (DFB) to occur. This bit is cleared automatically by the hardware when a DFB takes place. It must be reset with a Data Flag Register Bit Branch and Alter instruction or a Data Flag Register Load/Store instruction to re-enable the DFB. If bit 52 of word 0 is set the DFB takes place as described in 3.1.6.1 above if bit 51 of word 0 is also set. The order in which these two bits become set is immaterial; the result is an automatic DFB.

Bit 51 of word 0 is the dynamic inclusive OR of all product bits of word 0 and the individual products of bits 51 and 52 of words 1 through 3. It cannot be set or cleared directly by instruction, rather, the conditions causing it to be set must be cleared.

Bit 51 in each of words 1 through 3 is the dynamic inclusive OR of the product bits (00-15) of its respective word. In a similar manner, bit 52 in each of words 1 through 3 is a mask for bit 51 of its respective word. It is then the product of bit 51 and 52 of words 1 through 3 which is included in the dynamic inclusive OR for the automatic data flag branch. For words 1 through 3, bit 51 cannot be set or cleared directly by instruction, rather, the conditions causing it to be set must be cleared, while bit 52, on the other hand, <u>must</u> be set and cleared by instruction.

3.1.6.2.6 Conditional Inhibits

Bits 53 and 54 of words 2 and 3 provide the capability of selectively inhibiting the data flags of the same word from reflecting read keys in use, write keys in use, neither read keys nor write keys in use, or both read keys and write keys in use. Bit 53, when set, inhibits read dependencies from being reflected in the state of the data flags, and bit 54, when set, does the same for write dependencies. These bits are set and cleared only by execution of a Data Flag Branch Register Bit Branch and Alter instruction, or a Data Flag Register Load/Store instruction.

3.1.6.2.7 Data Flag Register Word 0 Bit Assignments

Product Field Bit / Mask Field Bit / Data Flag Bit or Free Flag Bit

Bits	Definitions -- Word 0
00-16-32	Undefined.
01-17-33	Undefined.
02-18-34	Undefined.
03-19-35	Soft Interrupt. Monitor software can set this data flag bit of a job's invisible package. If, after exchanging back to job mode, this data flag bit and this mask bit are set, a normal data flag branch occurs following completion of the current instruction.
04-20-36	Job Interval Timer.
05-21-37	Not Applicable.
06-22-38	Not Applicable.
07-23-39	The binary result exceeds the range of $\pm(2^{47}-1)$.
08-24-40	Bit 40 is the inclusive OR of bits 37, 38, and 39.

(continued)

3.1.6.2.7 (Cont.)

Product Field Bit / Mask Field Bit / Data Flag Bit or Free Flag Bit	Definitions -- Word 0 (continued)
09-25-41	Floating-point divide fault: The divisor has an all zero coefficient, or the divisor as read from the Register File or from Main Memory is machine zero. If the divisor and/or the dividend is indefinite, no divide fault exists. If a divisor causes a divide fault, the quotient is set to indefinite. The exponent overflow and result machine zero data faults are not set by a divide whose divisor caused a divide fault.
10-26-42	Exponent overflow: The exponent of the result is larger than $6FFF_{16}$ ($6F_{16}$ for 32-bit arithmetic). Results are not checked for exponent overflow until after the exponent adjustment for normalization or significance has taken place. In the adjust exponent instructions, if a left shift exceeds the number of places required for normalization, this data flag is set. Exponent overflow causes the result to be set to indefinite; therefore, the indefinite flag will always be set on an exponent overflow. This exponent oveflow data flag is not set if either source operand from Main Memory or the Register File is indefinite, or by a divide instruction whose divisor causes a divide fault.

(continued)

3.1.6.2.7 (Cont.)

Product Field Bit / Mask Field Bit / Data Flag Bit or Free Flag Bit	Definitions -- Word 0 (continued)
11-27-43	Result machine zero: The exponent of the result returned to Main Memory or to the Register File is less than 9000_{16} (90_{16} for 32-bit arithmetic). Result machine zero may be caused by exponent underflow, or by one or more of the input operands being machine zero. The result machine zero data flag bit is not set by a divide whose divisor causes a divide fault.
12-28-44	Bit 44 is the inclusive OR of bits 41, 42, and 43.
13-29-45	A negative source operand was encountered in a square root instruction. The square root of the absolute value of the operand is formed, and the two's complement of this square root is stored as the result.
14-30-46	An indefinite result was placed into Main Memory or into the Register File ...or... either or both operands of a floating-point compare were indefinite. An indefinite result may be caused by one or both operands of a floating-point arithmetic operation being indefinite, or by the occurence of either a divide fault or an exponent overflow.

(continued)

3.1.6.2.7 (Cont.)

```
 ---------- Product Field Bit
 |   ------- Mask Field Bit
 |   |   ---- Data Flag Bit or Free Flag Bit
 |   |   |  ---------------------------------------
 v   v   v  | Definitions -- Word 0 (continued)
-----------------------------------------------------
            |
 15-31-47   | Breakpoint:  See section 3.2.1.5.
            |
       48   | Undefined.
       49   | Undefined.
       50   | Undefined.
            |
       51   | This bit is the dynamic inclusive OR
            | of the product field bits of word 0
            | and the individual products of bits
            | 51 and 52 of words 1 through 3; it
            | is set if any one or more of these
            | conditions becomes set.  Bit 51
            | cannot be cleared directly; the
            | condition(s) causing it to be set
            | must be cleared to accomplish this.
            |
       52   | This bit is the data flag branch
            | enable; if it is a one and bit 51 of
            | word 0 becomes a one (or vice versa)
            | a data flag branch occurs at the end
            | of the current instruction.  See
            | 3.1.6.3 for additional information.
            | Bit 52 of word 0 is automatically
            | cleared by the execution of a data
            | flag branch.
            |
       53   | Not Applicable.
       54   | Not Applicable.
       55   | Not Applicable.
            |
```

(continued)

3.1.6.2.7 (Cont.)

```
 ---------- Product Field Bit
 |   ------- Mask Field Bit
 |   |   ---- Data Flag Bit or Free Flag Bit
 |   |   |  ------------------------------------
 v   v   v  | Definitions -- Word 0 (continued)
```

Bit	Definitions -- Word 0 (continued)
56	A CPU gate associated with the Maintenance Control Unit monitoring counters (See Functional Computer Specification 10354637).
57	A CPU gate associated with the Maintenance Control Unit monitoring counters (See Functional Computer Specification 10354637).
58	This bit set indicates that some other data flag bit has been set during execution of a scalar register instruction.
59	This bit set indicates that the breakpoint data flag (bit 47) was set by a breakpoint compare in Intermediate Memory (see section 3.2.1.5).
60	Undefined.
61	Undefined.
62	Undefined.
63	Undefined.

3.1.6.2.8 Data Flag Register Word 1 Bit Assignments

```
 ---------- Product Field Bit
 |  ------- Mask Field Bit
 |  |  ---- Data Flag Bit or Free Flag Bit
 |  |  |
 v  v  v  | Definitions -- Word 1
```

Bits (Product-Mask-Data/Free Flag)	Definitions -- Word 1
00-16-32	Undefined.
01-17-33	Interlock Flag 1.
02-18-34	Interlock Flag 2.
03-19-35	Interlock Flag 3.
04-20-36	Interlock Flag 4.
05-21-37	Interlock Flag 5.
06-22-38	Interlock Flag 6.
07-23-39	Interlock Flag 7.
08-24-40	Interlock Flag 8.
09-25-41	Interlock Flag 9.
10-26-42	Interlock Flag 10.
11-27-43	Interlock Flag 11.
12-28-44	Interlock Flag 12.
13-29-45	Interlock Flag 13.
14-30-46	Interlock Flag 14.
15-31-47	Interlock Flag 15.

(continued)

3.1.6.2.8 (Cont.)

```
 ---------- Product Field Bit
 |  ------- Mask Field Bit
 |  |  ---- Data Flag Bit or Free Flag Bit
 |  |  |  ----------------------------------
 v  v  v  | Definitions -- Word 1 (continued)
------------------------------------------------
          |
       48 | Undefined.
       49 | Undefined.
       50 | Undefined.
          |
       51 | This bit is the dynamic inclusive
          | OR of the product field bits of this
          | word (word 1); it is set if any one
          | or more of bits 00 through 15 of
          | this word are set.  This bit cannot
          | be cleared directly; bits 00 through
          | 15 of this word must be cleared
          | to accomplish this.
          |
       52 | This bit is a mask for bit 51 of
          | this word; it is the product of this
          | bit and bit 51 which is included in
          | the dynamic inclusive OR of bit 51,
          | word 0.  This bit must be set and
          | cleared by instruction.
          |
       55 | Undefined.
       56 | Undefined.
       57 | Undefined.
       58 | Undefined.
       59 | Undefined.
       60 | Undefined.
       61 | Undefined.
       62 | Undefined.
       63 | Undefined.
```

3.1.6.2.9 Data Flag Register Word 2 Bit Assignments

```
----------------------------------------------------------
| ---------- Product Field Bit                           |
| |  ------- Mask Field Bit                              |
| |  |  ---- Data Flag Bit or Free Flag Bit              |
| |  |  |  ---------------------------------------------|
| v  v  v | Definitions -- Word 2                        |
|--------------------------------------------------------|
|         |                                              |
| 00-16-32 | Undefined.                                  |
|          |                                             |
| 01-17-33 | Dependency Flag  1.                         |
|          |                                             |
| 02-18-34 | Dependency Flag  2.                         |
|          |                                             |
| 03-19-35 | Dependency Flag  3.                         |
|          |                                             |
| 04-20-36 | Dependency Flag  4.                         |
|          |                                             |
| 05-21-37 | Dependency Flag  5.                         |
|          |                                             |
| 06-22-38 | Dependency Flag  6.                         |
|          |                                             |
| 07-23-39 | Dependency Flag  7.                         |
|          |                                             |
| 08-24-40 | Dependency Flag  8.                         |
|          |                                             |
| 09-25-41 | Dependency Flag  9.                         |
|          |                                             |
| 10-26-42 | Dependency Flag 10.                         |
|          |                                             |
| 11-27-43 | Dependency Flag 11.                         |
|          |                                             |
| 12-28-44 | Dependency Flag 12.                         |
|          |                                             |
| 13-29-45 | Dependency Flag 13.                         |
|          |                                             |
| 14-30-46 | Dependency Flag 14.                         |
|          |                                             |
| 15-31-47 | Dependency Flag 15.                         |
----------------------------------------------------------
```

(continued)

3.1.6.2.9 (Cont.)

```
 ---------- Product Field Bit
 |  ------- Mask Field Bit
 |  |  ---- Data Flag Bit or Free Flag Bit
 |  |  |  ---------------------------------------
 v  v  v | Definitions -- Word 2 (continued)
```

Bit	Definitions -- Word 2 (continued)
48	Undefined.
49	Undefined.
50	Undefined.
51	This bit is the dynamic inclusive OR of the product field bits of this word (word 2); it is set if any one or more of bits 00 through 15 of this word are set. This bit cannot be cleared directly; bits 00 through 15 of this word must be cleared to accomplish this.
52	This bit is a mask for bit 51 of this word; it is the product of this bit and bit 51 which is included in the dynamic inclusive OR of bit 51, word 0. This bit must be set and cleared by instruction.
53	Inhibit dependency flags 1 through 15 for respective read key.
54	Inhibit dependency flags 1 through 15 for respective write key.
55	Undefined.
56	Undefined.
57	Undefined.
58	Undefined.
59	Undefined.
60	Undefined.
61	Undefined.
62	Undefined.
63	Undefined.

3.1.6.2.10 Data Flag Register Word 3 Bit Assignments

```
 ---------- Product Field Bit
 |   ------- Mask Field Bit
 |   |   ---- Data Flag Bit or Free Flag Bit
 |   |   |
 v   v   v    Definitions -- Word 3
```

Bits	Definitions -- Word 3
00-16-32	Dependency Flag 16.
01-17-33	Dependency Flag 17.
02-18-34	Dependency Flag 18.
03-19-35	Dependency Flag 19.
04-20-36	Dependency Flag 20.
05-21-37	Dependency Flag 21.
06-22-38	Dependency Flag 22.
07-23-39	Dependency Flag 23.
08-24-40	Dependency Flag 24.
09-25-41	Dependency Flag 25.
10-26-42	Dependency Flag 26.
11-27-43	Dependency Flag 27.
12-28-44	Dependency Flag 28.
13-29-45	Dependency Flag 29.
14-30-46	Dependency Flag 30.
15-31-47	Dependency Flag 31.

(continued)

3.1.6.2.10 (Cont.)

```
 ---------- Product Field Bit
 |   ------- Mask Field Bit
 |   |   ---- Data Flag Bit or Free Flag Bit
 |   |   |   ------------------------------------
 v   v   v  | Definitions -- Word 3 (continued)
```

Bit	Definitions -- Word 3 (continued)
48	Undefined.
49	Undefined.
50	Undefined.
51	This bit is the dynamic inclusive OR of the product field bits of this word (word 3); it is set if any one or more of bits 00 through 15 of this word are set. This bit cannot be cleared directly; bits 00 through 15 of this word must be cleared to accomplish this.
52	This bit is a mask for bit 51 of this word; it is the product of this bit and bit 51 which is included in the dynamic inclusive OR of bit 51, word 0. This bit must be set and cleared by instruction.
53	Inhibit dependency flags 16 through 31 for respective read key.
54	Inhibit dependency flags 16 through 31 for respective write key.
55	Undefined.
56	Undefined.
57	Undefined.
58	Undefined.
59	Undefined.
60	Undefined.
61	Undefined.
62	Undefined.
63	Undefined.

3.1.6.2.11 Data Flag Usage by Instruction

The following tables list all instructions by operation code in the first column. The other columns are headed with data flag bit numbers for word 0 of the data flag branch register. An X in one of these columns indicates that flag may be set during execution of that instruction. Some instructions affect no data flags while some may affect several.

Note that these are word 0 data flags; words 1, 2, and 3 contain data flags affected only by interlock and dependency keys.

(continued)

3.1.6.2.11 (Cont.)

OP CODE	WORD 0 DATA FLAG BITS 39	41	42	43	45	46	47	58	OP CODE	WORD 0 DATA FLAG BITS 39	41	42	43	45	46	47	58
00									20								
01									21								
02									22								
03									23								
04							X		24								
05									25								
06									26								
07									27								
08									28								
09									29								
0A									2A								
0B									2B								
0C									2C								
0D									2D								
0E									2E								
0F									2F								
10	X								30								
11									31								
12									32								
13									33								
14									34								
15									35								
16									36								
17									37								
18									38								
19									39								
1A									3A								
1B									3B								
1C									3C								
1D									3D								
1E									3E								
1F									3F								

(continued)

3.1.2.6.11 (Cont.)

OP CODE	WORD 0 DATA FLAG BITS 39	41	42	43	45	46	47	58	OP CODE	WORD 0 DATA FLAG BITS 39	41	42	43	45	46	47	58
40			X	X		X		X	60			X	X		X		X
41			X	X		X		X	61			X	X		X		X
42			X	X		X		X	62			X	X		X		X
43									63								
44			X	X		X		X	64			X	X		X		X
45			X	X		X		X	65			X	X		X		X
46			X	X		X		X	66			X	X		X		X
47									67								
48			X	X		X		X	68			X	X		X		X
49			X	X		X		X	69			X	X		X		X
4A									6A								
4B			X	X		X		X	6B			X	X		X		X
4C		X	X	X		X		X	6C		X	X	X		X		X
4D									6D								
4E									6E								
4F		X	X	X		X		X	6F		X	X	X		X		X
50						X		X	70						X		X
51						X		X	71						X		X
52						X		X	72						X		X
53				X	X	X		X	73				X	X	X		X
54			X	X		X		X	74			X	X		X		X
55			X			X		X	75			X			X		X
56									76			X	X		X		X
57									77			X	X		X		X
58									78								
59			X	X		X		X	79			X	X		X		X
5A									7A								
5B									7B								
5C				X		X		X	7C								
5D				X		X		X	7D								
5E									7E								
5F									7F								

(continued)

3.1.6.2.11 (Cont.)

OP CODE	WORD 0 DATA FLAG BITS 39	41	42	43	45	46	47	58
80								
81								
82								
83								
84								
85								
86								
87								
88								
89								
8A								
8B								
8C								
8D								
8E								
8F								
90								
91								
92								
93								
94								
95								
96								
97								
98								
99								
9A								
9B								
9C								
9D								
9E								
9F		X	X	X		X		

OP CODE	WORD 0 DATA FLAG BITS 39	41	42	43	45	46	47	58
A0								
A1								
A2								
A3								
A4								
A5								
A6								
A7								
A8								
A9								
AA								
AB								
AC								
AD								
AE								
AF								
B0						X		X
B1						X		X
B2						X		X
B3						X		X
B4						X		X
B5						X		X
B6								
B7								
B8								
B9								
BA								
BB								
BC								
BD								
BE								
BF								

(continued)

3.1.6.2.11 (Cont.)

OP CODE	WORD 0 DATA FLAG BITS 39	41	42	43	45	46	47	58	OP CODE	WORD 0 DATA FLAG BITS 39	41	42	43	45	46	47	58
C0									E0								
C1									E1								
C2									E2								
C3									E3								
C4									E4								
C5									E5								
C6									E6								
C7									E7								
C8									E8								
C9									E9								
CA									EA								
CB									EB								
CC									EC								
CD									ED								
CE									EE								
CF									EF								
D0									F0								
D1									F1								
D2									F2								
D3									F3								
D4									F4								
D5									F5								
D6									F6								
D7									F7								
D8									F8								
D9									F9								
DA									FA								
DB									FB								
DC									FC								
DD									FD								
DE									FE								
DF									FF								

3.1.6.3 Data Flag Branch (DFB) [A6.0]

If a bit in the mask field is set and its associated masked data flag bit is set, the associated bit in the product field becomes a one. Bit 51 in the free flag field also becomes a one since it is the dynamic inclusive OR of bits 4 through 15 of the product field.

If bit 51 is a one from above and if bit 52 is also set (this is the DFB enable bit), an automatic DFB occurs. The DFB takes place sometime following the termination of the instruction which caused the DFB condition to exist. The execution of the DFB sets the bit address of the next instruction into the right-most 48 bits of register 01 and a branch is made to the bit address contained in the right-most 48 bits of register 02. The DFB enable bit in the flag mask register (bit 52) is automatically cleared at this time. The left-most 16 bits of register 01 are cleared to zero by a DFB.

Programmer Note:

DFBs are disabled when bit 52 is cleared. But if bit 52 is reset before eliminating all the DFB conditions, another DFB will occur which will change the return address in register 01 and the machine may wind up in a "tight loop" if proper caution is not taken. Sampling bit 51 for a zero before setting bit 52 will prevent this situation for all cases except those involving the job interval timer. When using the job interval timer, it should be remembered that the setting of bit 36 in the DFR occurs asynchronously with respect to instruction execution once the job interval timer is loaded. Thus the time may set bit 36 after the check of bit 51 and before the branch to the contents of register 01. One method of handling this situation is to examine the contents of register 01 upon entering the routine for handling data flag branches. If register 01 indicates that the branch occurred outside the DFB routine, then register 01 could be copied to a temporary location.

If register 01 indicated that the branch had occurred within the DFB routine, then register 01 would not be copied to the temporary location. At the conclusion of the DFB routine, a branch would always be taken to the contents of the temporary location.

(continued)

3.1.6.3 (Cont.)

A simpler method is to combine the setting of bit 52 and the branch to the contents of register 01 into a single 33 instruction (33603401).

3.1.7 Register File

For register operations, the 8-bit instruction designators directly address the 256_{10} registers of the Register File. During program execution (monitor or job), these registers reside in the Register File. When an exchange operation occurs, the registers are stored into 256_{10} memory locations beginning at bit address zero if in monitor mode and bit address 4000_{16} if in job mode. The registers may not be referenced as memory by their associated monitor or job program.

Figure 1 shows a map of the Register File and the relationship between the register, its memory address for monitor mode and its 8-bit designator. The number on the right represents the bit address and the number on the left is the value of the 8-bit designator for the 64-bit register case. The number inside the register represents the value of the 8-bit designator for the 32-bit operand case. Note that any reference to 32-bit register one is undefined.

(continued)

3.1.7 (Cont.)

<u>8-bit Designator</u> <u>Monitor Mode Bit Address</u>

```
         Bit
         0                31 32                63
        --------------------------------------------
    0  |        0          |/////////////////|0...0000
       |-------------------|-----------------|       16
    1  |        2          |        3        |0...0040
       |-------------------|-----------------|        16
    2  |        4          |        5        |0...0080
       |-------------------------------------|        16
       \                                     /

       /                                     \
       |-------------------------------------|
   7F  |      FE16         |       FF16      |0...1FC0
       |-------------------------------------|        16
   80  |                                     |0...2000
       |-------------------------------------|        16
       \                                     /

       /                                     \
       |                                     |
  FF   |-------------------------------------|
   16  |                                     |0...3FC0
        --------------------------------------        16
```

Figure 1. Register File

<u>Register File Restrictions</u>

A. Register Zero (Job or Monitor Mode)

1. During an exchange operation the contents of the trace register and the appropriate memory location for register zero are exchanged (swapped).

Monitor to Job:

	Before Exchange	After Exchange
Absolute Address Zero	A	C
Trace Register	C	A

(continued)

3.1.7 (Cont.)

Job to Monitor:

	Before Exchange	After Exchange
Absolute Address Zero	A	A
Trace Register	C	A

During a 7D (Swap) instruction involving register zero as part of the register field, note a required peculiarity. Although the current contents of the trace register are sent to the appropriate memory location for register zero, the current contents of the trace register are not altered.

	Contents Before 7D	Contents After 7D
Memory location for register zero	A	B
Trace register	B	B

2. Register zero when referenced by a designator will provide machine zero as an operand except when used as a source register for a base address or other description for a stream instruction, in which case register zero will appear to contain 64-zero bits. The use of a zero address may cause the instruction to be treated as an illegal instruction. If register zero is specified as the destination register, the instruction typically performs normally with data flags being set, if warranted, but no data is stored. Some instructions become undefined if register zero is specified as a destination register.

(continued)

3.1.7 (Cont.)

The following tables are intended to define what operand is obtained when register zero is specified for a source operand. To simplify this chart, specifying of register zero as a destination register has been ignored. A blank in the chart indicates where it is either not possible to specify register zero or it may only be specified as a destination register. The designators R, S, T, U, V, W, G, X, A, Y, B, Z, and C are used for convenience although they do not apply to all instructions. Utilization of the following symbols is made.

Symbol	Result When Register Zero is Referenced for an Operand
M	Machine zero is provided. $8000\ 0000\ 0000\ 0000_{16}$ 64-bit mode $8000\ 0000_{16}$ 32-bit mode
A	All zero is provided.
Z	All zero in the used portion. In this instance the left-most bit is not used thus machine zero and all zeros are indistinguishable.
N	Instruction performs as a No op.
C	No control vector is used.
O	A mask of all ones is provided.

(continued)

3.1.7 (Cont.)

Op Code	Instruction Designator R	S	T
04	Z		
09		Z	Z
0A	Z		
0E	Z	Z	
10	M		
11	Z		
12	Z	Z	
13	Z	Z	Z

Op Code	Instruction Designator R	S	T
24	Z	Z	
25	Z	Z	M
26	Z	Z	
27	Z	Z	M
2B	M	Z	
2C	M	M	
2D	M	M	
2E	M	M	
2F		Z	Z
30	M		
31	Z	Z	Z
32		Z	Z
33			Z
34	M	Z	
35	Z	Z	Z
36		Z	Z
37			
38	M		
3A	Z		
3B	Z		
3C	Z	Z	
3D	Z	Z	
3F	Z		

(continued)

3.1.7 (Cont.)

Op Code	Instruction Designators R	S	T
40	M	M	
41	M	M	
42	M	M	
44	M	M	
45	M	M	
46	M	M	
48	M	M	
49	M	M	
4B	M	M	
4C	M	M	
4D			
4E	Z		
4F	M	M	
50	M		
51	M		
52	M		
53	M		
54	M	Z	
55	M	M	
58	M		
59	M		
5A	M		
5B	Z	Z	
5C	M		
5D	M		
5E	Z	Z	
5F	Z	Z	M

Op Code	Instruction Designators R	S	T
60	M	M	
61	M	M	
62	M	M	
63	M	Z	
64	M	M	
65	M	M	
66	M	M	
67	M	Z	
68	M	M	
69	M	M	
6B	M	M	
6C	M	M	
6D	M	Z	
6E	M	Z	
6F	M	M	
70	M		
71	M		
72	M		
73	M		
74	M	Z	
75	M	Z	
76	M		
77	M		
78	M		
79	M		
7A	M		
7B	Z	Z	
7C	M		
7D	A	*	A
7E	Z	Z	
7F	Z	Z	M

*See Section 3.1.7.A.1 above

(continued)

3.1.7 (Cont.)

Op Code	Instruction Designators T	U	V	W
9D	A	A	A	A
9E	A	A	A	A
9F			A	A

Op Code	G	X	Instruction Designators A	Y	B	Z	C
B0		Z	M	Z	Z	Z	
B1		Z	M	Z	Z	Z	
B2		Z	M	Z	Z	Z	
B3		Z	M	Z	Z	Z	
B4		Z	M	Z	Z	Z	
B5		Z	M	Z	Z	Z	
B6	Z						

(continued)

3.1.7 (Cont.)

B. 64-bit registers one and two (32-bit registers 2 through 5)

If data flag branches are being used, 64-bit registers one and two must be reserved exclusively for that use. Register one is the data flag branch exit address and register two holds the data flag branch entry address.

C. Monitor's 64-bit registers $0\text{-}F_{16}$ (32-bit registers $0\text{-}1F_{16}$)

Registers zero, one and two have the restrictions listed in A and B above. Registers 3 through 7 are used for the illegal instruction, exit force, and external interrupt entry points.

D. 32-bit register one (right-most half of 64-bit register 0)

Any reference to 32-bit register one is undefined.

3.1.8 Real-Time Counters

3.1.8.1 Free-Running Clock

This clock consists of a free-running 47-bit counter and a positive sign bit for a total of 48 bits. It can be stored into register T using a Transmit Real-Time Clock to T (39) instruction. This counter increments at a one MHz rate.

3.1.8.2 Monitor Interval Timer

The monitor interval timer is a 24-bit timer that decrements at a one MHz rate.

This timer can be loaded from register R using the Transmit (R) to Monitor Interval Timer (0A) instruction, when the computer is in monitor mode. The timer can be activated by loading it with anything but all zeros. Once it is activated, it will decrement until it reaches zero or is deactivated. When the timer is decremented to zero, it will cause an external interrupt 16 which must be processed like any other external interrupt.

The timer is deactivated by the following methods:

1. Master clear
2. Loading with all zeros
3. Decremented to all zeros (when it is decremented to all zeros and caused an external interrupt, it will be inactive until loaded with some value other than zero).

3.1.8.3 Job Interval Timer

The job interval timer is a 24-bit counter decrementing at a one MHz rate.

This clock can be loaded (in job mode) only from register R using a 3A (Transmit (R) to Job Interval Timer) instruction. Once loaded, the timer continues to decrement until either an exchange to monitor mode occurs, the timer decrements to zero, or the timer is loaded with a value of zero. If an exchange to monitor mode occurs, the decrementing of the job interval timer is stopped and the current contents of the timer are stored in the invisible package. When the execution of that job is resumed, the job interval timer is loaded from the invisible package and resumes decrementing.

(continued)

3.1.8.3 (Cont.)

When the timer decrements to zero, bit 36 of the data flag branch register will be set. Thus, if the corresponding mask bit is set, a data flag branch would then occur during the next RNI.

The timer may be deactivated by loading it with a value of zero. This does not cause bit 36 of the data flag branch register to be set. Master clear will also deactivate the job interval timer.

The timer is deactivated by the following methods:

1. Master clear
2. Loading with a value of zero
3. Decrementing to zero

The contents of the job interval timer may be sampled by use of the 37 instruction (Transmit Job Interval Timer to T). This does not deactivate the counter.

3.1.9 (N/A)

3.1.10 Exchange Operations and Invisible Package

The purpose of the exchange is to change the prime role of the CPU from monitor mode to job mode or from job mode to monitor mode.

The exchange operation from monitor to a job is always accomplished with an Exit Force instruction. This causes the contents of the invisible package to be loaded into the appropriate registers; the mode to be changed from monitor to job enabling interrupts; and execution to begin as specified by the invisible package. Note that this may be the restarting of a previously interrupted program.

(continued)

3.1.10 (Cont.)

The Exit Force instruction and the channel interrupt are the two normal ways of getting from a job in job mode to the monitor program in monitor mode. Attempting to execute a monitor-type instruction in job mode or by attempting to execute an undefined op-code comprise the third way into the monitor. Except for the starting point in the monitor program, the operation performed in getting to the monitor are identical for the three. Sufficient information to restart this job is stored into the invisible package and the mode is changed from job to monitor. The monitor program is executed starting at the absolute address contained in the right-most 48 bits of the monitor's register 3, 5, or 6.

Method of getting to the Monitor	Monitor register, the contents of which is used to set P
1. Attempt to perform an illegal instruction or a monitor-type instruction in job mode	Register 3
2. Attempt to perform an illegal instruction in monitor mode	Register 4
3. Exit force	Register 5
4. External interrupt	Register 6

The monitor must set up the invisible package for each job. There is no invisible package for the monitor program itself.

To start a job initially, the monitor must clear the entire invisible package area except for the program address areas.

For a more detailed description of the exchange operation and the invisible package, see the applicable computer specification as listed in Section 2.0.

3.2 Performance Characteristics

3.2.1 Instruction Descriptions

The instruction titles (3.2.1.1 - 3.2.1.256) are written in the following format:

3.2.1.XXX AA B CC DD NAME OF INSTRUCTION [AX]

where AA = the function code ($00\text{-}FF_{16}$)

B = the format types, 1-F

CC = the number of bits in the operand

1	single bit
8	bytes
32	half-words
64	words
E	either 32 or 64-bit
B	both 32 and 64-bit
NA	operand size not applicable

DD = the instruction type

Blank	Undefined
BR	Branch
IN	Index
MN	Monitor
NT	Non-Typical
RG	Register
SM	Stream

[AX] = The section in the Appendix which gives further information.

In the instruction descriptions which follow, all data flags referred to are in word 0 of the data flag branch register.

3.2.1.1 00 4 NA MN IDLE

When in monitor mode, enable the external interrupts and idle until an external interrupt occurs. The R, S, and T designators are undefined and must be set to zero.

3.2.1.2 01 ILLEGAL

3.2.1.3 02 ILLEGAL

3.2.1.4 03 ILLEGAL

3.2.1.5 04 4 64 NT BREAKPOINT-MAINTENANCE

The breakpoint instruction transfers (R) to the main memory breakpoint register and (S) to the intermediate memory breakpoint register. The breakpoint registers are used as maintenance and program debugging aids. If R or S is zero, the respective breakpoint register is not altered. The T field is not used.

Register R and S Format

Usage Bits	Breakpoint Address
0 15	16 63

Breakpoint Compares

1. Main memory addresses are compared with the main memory breakpoint register and intermediate memory addresses are compared with the intermediate memory breakpoint register, both according to the respective usage bits.

2. Since the monitor program does not have an invisible package, the breakpoint registers must be set up each time the monitor program is entered. The breakpoint registers are automatically cleared to zero during the exchange to the monitor.

3. Program address compares are made on half-word boundaries, and all other compares are made on sword boundaries; thus bits 59-63 are not used for program address compares, and bits 55-63 are not used for all other compares.

(continued)

3.2.1.5 (Cont.)

The two breakpoint address registers are compared with all addresses delivered to their respective memory. If the breakpoint address matches one of these addresses and the proper usage bit is set, bit 47 of the data flag branch register is set indicating a breakpoint. In addition, if bit 47 is set as the result of a match in Intermediate Memory, bit 59 is also set. Any combination of usage bits is permissible; therefore the breakpoint address can be checked against any or all of the addresses listed below. The breakpoint registers are part of the invisible package of a job.

Main Memory Breakpoint

Bits 3-15 of the register designated by R provide the usage bits for main memory breakpoint address compares; bits 0-2 are undefined and must be set to zero. For each of the following bits set, the indicated address being supplied to Main Memory is compared to the breakpoint address.

Bit 3 - VR1 read operand address
Bit 4 - VR2 read operand address
Bit 5 - VR3 read operand address
Bit 6 - VR4 read operand address
Bit 7 - VR1 write operand address
Bit 8 - VR2 write operand address
Bit 9 - MR1 read operand address
Bit 10 - MR2 read operand address
Bit 11 - MR3 read operand address
Bit 12 - MW1 write operand address
Bit 13 - RD scalar read operand address
Bit 14 - WT scalar write operand address
Bit 15 - half-word program address (P) just after execution of the instruction at that address.

Intermediate Memory Breakpoint

Bits 4-15 of the register designated by S provide the usage bits for intermediate memory breakpoint address compares; bits 0-3 are undefined and must be set to zero. For each of the following bits set, the indicated address being supplied to Intermediate Memory is compared to the breakpoint address.

(continued)

3.2.1.5 (Cont.)

Bit 4 - RW1 read operand address
Bit 5 - RW2 read operand address
Bit 6 - RW3 read operand address
Bit 7 - RW4 read operand address
Bit 8 - RW1 write operand address
Bit 9 - RW2 write operand address
Bit 10 - RW3 write operand address
Bit 11 - RW4 write operand address
Bit 12 - Low speed ports 0-3 read operand address
Bit 13 - Low speed ports 4-7 read operand address
Bit 14 - Low speed ports 0-3 write operand address
Bit 15 - Low speed ports 4-7 write operand address

Data flags: bits 47 and 59

3.2.1.6 05 ILLEGAL

3.2.1.7 06 7 NA MN FAULT TEST - MAINTENANCE

This instruction is used to complement checkword bits on the scalar write buses to Main Memory and to Intermediate Memory in order that the read SECDED circuitry may be checked. It can also be used to disable the error correction circuitry on all read buses. This allows data to be passed through the SECDED hardware without any correction taking place.

This instruction is always enabled during monitor mode. In job mode it becomes a No op unless bit 13 of word 8 in the job's invisible package is set.

The modes are set up by executing this instruction with a "1" in the appropriate R or S designator bits and are cleared by executing the instruction with a "0" in the same bit locations. No interrupts or I/O requests to Intermediate Memory can be allowed during these fault tests.

The R and S designator bits are defined below; the T designator is unused and must be set to zero.

(continued)

3.2.1.7 (Cont.)

R DESIGNATOR BIT

8	Disable error correction on all read buses.
9-15	Checkword bits to be complemented on scalar write to Main Memory.

S DESIGNATOR BIT

16-23	Checkword bits to be complemented on scalar write to Intermediate Memory.

Programmer Note: These bits must be set to zero before any monitor to job exchange operation. If these bits are not set to zero via the 06 instruction, the connection network could produce invalid data on the read and invalid data could be written into memory.

A description of each of these faults can be found in specification 10354637, CDC FMP Functional Computer Specification.

SECDED FAULTS

The test is initiated by executing an 06 instruction with bits 9 through 23 selected (R and S designators to complement the respective checkword bits of) half-words 0, 1, 2, and 3 on the scalar write bus to Main Memory, and/or full words on the scalar write bus to Intermediate Memory. By appropriate selectio of data bits and complementation of checkword bits n when writing in memory, it is possible to generate SECDED faults on all read buses. This should allow complete checking of the read SECDED hardware and also the fault recording hardware for type and address of the fault.

The forced complementing of the checkword bits is discontinued by executing the 06 instruction with bits 9 through 15 (R designator) and/or bits 16-23 (S designator) set to zero.

3.2.1.8 07 ILLEGAL

3.2.1.9 08 4 NA MN INPUT/OUTPUT PER R

When in monitor mode: Activate the channel flag designated by the R designator and exit to the next sequential instruction. If the R designator specifies a non-existent channel, the operation of this instruction is undefined.

The S and T designators are undefined and must be set to zero.

3.2.1.10 09 4 64 BR EXIT FORCE

From a job to the monitor: Exchange to the monitor program. A hardware branch is then taken to the address defined by the right-most 48 bits of the monitor's register 5.

From the monitor to a job: Exchange to the job whose invisible package is located starting at the absolute bit address 104000_{16}.

The R, S, and T designators are undefined and must be set to zero.

The exchange operation and invisible package are explained in section 3.1.10; also, additional information is provided in the computer specification listed in Section 2.0.

3.2.1.11 0A 4 64 MN TRANSMIT (R) TO MONITOR INTERVAL TIMER

When in monitor mode, transmit bits 40 through 63 of 64-bit register R to the monitor interval timer (see Section 3.1.8). The left-most 40 bits of register R are ignored. The S and T designators are undefined and must be set to zero.

3.2.1.12 0B ILLEGAL

3.2.1.13 0C ILLEGAL

3.2.1.14 0D ILLEGAL

3.2.1.15 OE 4 64 MN TRANSLATE EXTERNAL INTERRUPT

Each bit in the external interrupt register (EIR) is associated with an external I/O channel, the Swap Unit, or the monitor interval timer.

External Interrupt Register Bit	Assignment
0	I/O Channel 0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	I/O Channel 13
14	Swap Unit
15	Monitor Interval Timer

Translate the lowest numbered bit set in the EIR into its associated four-bit code and transmit this code to the right-most four bits of register T. The left-most 60 bits of register T are cleared to zero.

Examine the EIR and if only one bit is set, the branch condition is met. The branch, if taken, is to (S) + (R) where (S) is an index in half-words and (R) is the base address.

The exit, be it a branch or not, clears the bit (and only that bit) in the EIR corresponding to the channel designator which was transmitted to register T.

If the T and S designators are equal, the interrupting channel designator will also be the branch index.

Bit zero of the EIR will never be set as it is reserved for maintenance purposes.

If no bit in the EIR is set, this instruction sets T to all zeros and no branch is taken.

3.2.1.16 OF ILLEGAL

3.2.1.17 10 A 64 RG CONVERT BCD TO BINARY, FIXED LENGTH

Convert the packed BCD number in register R to a signed (two's complement) binary number and place the result into the right-most 48 bits of register T. The conversion is undefined for binary results greater than $2^{47}-1$ or less than negative $(2^{47}-1)$; thus the largest decimal number that may be converted is $\pm$140,737,488,355,327. The ASCII/EBCDIC sign code for the BCD number is in bits 60-63 of register R.

Data flag bit 39 will be set for numbers outside this range.

If the input number is not a valid BCD number, the results are undefined. Bits 0-15 of register T will be cleared to zero.

Data flag: bit 39

3.2.1.18 11 A 64 RG CONVERT BINARY TO BCD, FIXED LENGTH

Convert the right-most 48 bits (two's complement binary number) of register R to a packed BCD number and place the result in register T. The result is a number having 15 digits (4 bits per digit plus the sign in th lower bits - bits 60-63). The binary range is $\pm(2^{47}-1)$. During job mode, the sign bits generated are conditioned by the ASCII/EBCDIC bit in the job's invisible package. During monitor mode, only ASCII codes will be generated.

3.2.1.19 12 7 8 NT LOAD BYTE; (T) PER (S), (R)
3.2.1.20 13 7 8 NT STORE BYTE; (T) PER (S), (R)

Load/store a byte from/into the address specified by (R) + (S), where (R) is the base address and (S) is an item count of bytes, into/from bits 56 through 63 of register T. The remaining bits of T are cleared on a load and ignored on a store.

3.2.1.21 14 ILLEGAL

3.2.1.22 15 ILLEGAL

3.2.1.23 16 ILLEGAL

3.2.1.24 17 ILLEGAL

3.2.1.25 18 ILLEGAL

3.2.1.26 19 ILLEGAL

3.2.1.27 1A ILLEGAL

3.2.1.28 1B ILLEGAL

3.2.1.29 1C ILLEGAL

3.2.1.30 1D ILLEGAL

3.2.1.31 1E ILLEGAL

3.2.1.32 1F ILLEGAL

3.2.1.33 20 7 64 RG SHIFT (R) AND (R+1) PER S TO (T) AND (T+1)

This instruction shifts the 128-bit operand formed by catenating the contents of register R and register R+1 (bit 0 of register R+1 follows bit 63 of register R) and stores the results into the register designated by T and the next sequential register (T+1). The S designator specifies the type and amount of shift. If the S designator is in the range from 0 through $7F_{16}$ (0 through 127_{10}), the 128-bit operand is shifted left end-around the specified number of places. If the S designator is in the range from FF through 81_{16} (-1 through -127_{10}), the 128-bit operand is shifted right with sign extension. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the S designator. If for example, S is equal to FE_{16}, the operand shifts right two places. If the S designator is greater that $7F_{16}$ or less than 81_{16}, the results of this instruction are undefined. The R designator must specify an even register number. If the R designator is equal to zero, register zero will provide machine zero. This instruction does not test for machine zero or indefinite or set any data flags.

3.2.1.34 21 7 64 RG SHIFT (R) AND (R+1) PER (S) TO (T) AND (T+1)

This instruction shifts the 128-bit operand formed by catenating the contents of register R and register R+1 (bit 0 of register R+1 follows bit 63 of register R) and stores the results into the register designated by T and the next sequential register (T+1). The contents of the register designated by S determine the type and amount of shift. If the

(continued)

3.2.1.34 (Cont.)

right-most byte of register S is in the range from 0 through $7F_{16}$ (0 through 127_{10}), the 128-bit operand is shifted left end-around the specified number of places. If the right-most byte of register S is in the range from FF_{16} through 81_{16} (-1 through -127_{10}), the 128-bit operand is shifted right with sign extension. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the right-most byte of register S. If the right-most byte of register S is greater than $7F_{16}$ or less than 81_{16}, the results of this instruction are undefined. The left-most seven bytes of register S are ignored.

The R designator must specify an even register number. If the R designator is equal to zero, register zero will provide machine zero. This instruction does not cause a test for machine zero or indefinite or set any data flags.

3.2.1.35 22 ILLEGAL

3.2.1.36 23 ILLEGAL

3.2.1.37 24 7 32 NT INTERMEDIATE MEMORY LOAD; (T) PER (S), (R)

3.2.1.38 25 7 32 NT INTERMEDIATE MEMORY STORE; (T) PER (S), (R)

Load/store 32-bit register T from/into the Intermediate Memory address specified by (R) + (S), where (R) is the base address and (S) is an item count of half-words. Note that R and S are 64-bit registers and that the item count is shifted left five places before the addition. Overflow from this addition is ignored, if it occurs.

3.2.1.39 26 7 64 NT INTERMEDIATE MEMORY LOAD; ($\bar{T}$) PER (S), (R)

3.2.1.40 27 7 64 NT INTERMEDIATE MEMORY STORE; (T) PER (S), (R)

Load/store 64-bit register T from/into the Intermediate Memory address specified by (R) + (S), where (R) is the base address and (S) is an item count of words. Note that the item count is shifted left six places before the addition. Overflow from this addition is ignored, if it occurs.

3.2.1.41 28 ILLEGAL

3.2.1.42 29 ILLEGAL

3.2.1.43 2A ILLEGAL

3.2.1.44 2B 4 64 RG ADD TO LENGTH FIELD

Add bits 00 through 15 of register R to bits 48 through 63 of S and store the result in bits 00 through 15 of register T. Bits 16 through 63 of register R are transferred to bits 16 through 63 of register T.

3.2.1.45 2C 4 64 RG LOGICAL EXCLUSIVE OR (R),(S), TO (T)
3.2.1.46 2D 4 64 RG LOGICAL AND (R),(S), TO (T)
3.2.1.47 2E 4 64 RG LOGICAL INCLUSIVE OR (R),(S), TO (T)

These instructions perform the indicated logical functions listed below. The function occurs bit by bit on the 64-bit operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

R	S	EXCLUSIVE OR R,S	AND R,S	INCLUSIVE OR R,S
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	1

If the R or S designators equal zero, register zero will contain machine zero.

3.2.1.48 2F 9 1 BR REGISTER BIT BRANCH AND ALTER

This instruction examines bit 63 of register T. As specified by the G designator a branch is made to the address contained in the right-most 48 bits of register S. The branch is made according to G bits 0 and 1 as follows:

G0	G1	
0	0	do not branch.
0	1	branch unconditionally.
1	0	branch if the object bit was a one.
1	1	branch if the object bit was a zero.

After the branch decision has been made and regardless of what that decision was, the object bit is altered according to G bits 2 and 3 as follows:

G2	G3	
0	0	do not alter the object bit.
0	1	toggle the object bit to the other state.
1	0	set the object bit to a one.
1	1	clear the object bit to a zero.

3.2.1.49 30 7 64 RG SHIFT (R) PER S TO (T)

This instruction shifts the 64-bit operand from the register designated by R and stores the result into the register designated by T. The S designator specifies the type and amount of the shift. If the S designator is in the range from 0 through $3F_{16}$ (0 through 63_{10}), the operand from register R is shifted left end-around the specified number of places and then stored in register T. If the S designator is in the range from FF_{16} through $C1_{16}$ (-1 through -63_{10}), the operand from register T is shifted right with sign extension and then stored into register T. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the S designator. If for example, S is equal to FE_{16}, the operand from register R shifts right two places. If the S designator is greater than $3F_{16}$ or less than $C1_{16}$, the results of this instruction are undefined.

If the R designator is equal to zero, register zero will provide machine zero. This instruction does not test for machine zero or indefinite or set any data flags.

3.2.1.50 31 7 64 BR INCREASE(R) AND BRANCH IF(R) <> 0

Increment the contents of the right-most 48 bits of register R by one. The upper 16 bits of register R are not altered and arithmetic overflow is ignored.

If the result from above is 48 zeros, go to the next sequential instruction. If the 48-bit result from above is non-zero, branch to (S) + (T) where (S) is an item count of half-words and (T) is the base address. The resulting address for the branch is undefined if the R designator is equal to either the S designator or the T designator.

3.2.1.51 32 9 1 BR BIT BRANCH AND ALTER

Register S contains the address of the object bit. This instruction reads up the word containing the object bit and examines the bit. The branch is then made according to G bits 0 and 1:

G0	G1	
0	0	do not branch.
0	1	branch unconditionally.
1	0	branch if the object bit was a one.
1	1	branch if the object bit was a zero.

After the branch decision has been made and regardless of what that decision was, the object bit is altered according to G bits 2 and 3 as follows:

G2	G3	
0	0	do not alter the object bit.
0	1	toggle the object bit to the other state.
1	0	set the object bit to a one.
1	1	clear the object bit to a zero.

NOTE: If G0 and G2 and G3 = 0, do not reference the object bit at all.

If (G0 = 1) and (G2 and G3 = 0) read, but do not write the object bit.

G bit 5 = 0 Register T contains the branch address.

G bit 5 = 1, G bit 6 = 0 — Branch address is formed by adding the T designator, used as a half-word item count, to the program address register.

G bit 5 = 1, G bit 6 = 1 — Branch address is formed by subtracting the T designator, used as a half-word item count, from the program address register.

3.2.1.52 33 B 1 BR DATA FLAG REGISTER BIT BRANCH AND ALTER

I is a six-bit designator specifying an object bit in word N of the data flag register. The object bit in the data flag register is examined and the decision to branch is made according to G bits 0 and 1.

G0	G1	
0	0	do not branch.
0	1	branch unconditionally.
1	0	branch if the object bit was a one.
1	1	branch if the object bit was a zero.

After the branch decision has been made and regardless of what that decision was, the object bit is altered according to G bits 2 and 3 as follows:

G2	G3	
0	0	do not alter the object bit.
0	1	toggle the object bit to the other state.
1	0	set the object bit to a one.
1	1	clear the object bit to a zero.

Programmer Note: It is meaningless to try to alter bits in the product field (bits 0-15) of word 0 since the word 0 product field is strictly a function of the appropriate data flag and flag mask bits. For words 1 through 3, on the other hand, data flags (bits 32-47) cannot be altered since they are strictly a function of the state of interlock key and dependency key usage. Conversely, the product bits of words 1 through 3 must be cleared by instruction and can be cleared only (cannot be set).

Since the 33 instruction begins execution without waiting until the machine has completed all operations, the data flag bits may set on any minor cycle during execution of the 33 instruction. Therefore, the object bit is sampled 2 minor cycles after the 33 instruction is loaded into IRO. This sampled object bit, rather than the actual object bit, is used to control the decision to branch, and the altering of the actual object bit in the data flag register. Consequently, any data flag bits that set after the object bit is sampled will not

(continued)

3.2.1.52 (Cont.)

affect the decision to branch. Also, if the sampled object bit is a zero, any data flag bits that set afterwards will not be cleared nor toggled to a zero.

G bit 5 = 0	Register T contains the branch address.
G bit 5 = 1, G bit 6 = 0	Branch address is formed by adding the T designator, used as a half-word item count, to the program address register.
G bit 5 = 1, G bit 6 = 1	Branch address is formed by subtracting the T designator, used as a half-word item count, from the program address register.

3.2.1.53 34 4 64 RG SHIFT(R) PER (S) TO (T)

This instruction shifts the 64-bit operand from the register designated by R and stores the result into the register designated by T. The register designated by S specifies the type and amount of the shift. If the right-most byte of register S is in the range from 0 through $3F_{16}$ (0 through 63_{10}), the shift is left, end-around the specified number of places. If the right-most byte of register S is in the range from FF_{16} through $C1_{16}$ (-1 through -63_{10}), the shift is right with sign extension. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the right-most byte of register S. If the right-most byte of register S is greater than $3F_{16}$ or less than $C1_{16}$, the results of this instruction are undefined. The left-most seven bytes of register S are ignored.

If the R designator is equal to zero, register zero will provide machine zero. This instruction does not cause a test for machine zero or indefinite or set any data flags.

3.2.1.54 35 7 64 BR DECREASE (R) AND BRANCH IF (R) <> 0

Decrement the contents of the right-most 48 bits of register R by one. The upper 16 bits of register R are not altered and arithmetic overflow is ignored.

If the result from above is 48 zeros, go to the next sequential instruction. If the 48-bit result from above is non-zero, branch to (S) + (T) where (S) is an item count of half-words and (T) is the base address. The resulting address for the branch is undefined if the R designator is equal to either the S designator or the T designator.

3.2.1.55 36 7 64 BR BRANCH AND SET (R) TO NEXT INSTRUCTION

After storing the address of the next sequential instruction into register R, branch to (S) + (T) where (S) is an item count of half-words and (T) is the base address. Bits 0 through 15 of register R are forced to zeros. Bits 59 through 63 of register R are undefined. If the R designator is equal to the S designator the results of this instruction are undefined.

NOTE: If S=0, and R=T, this instruction sets register R to the half-word address of the next instruction and the program continues at the next instruction. This is a way to sample the program address register (P).

3.2.1.56 37 A 64 NT TRANSMIT JOB INTERVAL TIMER TO (T)

Transmit the contents of the job interval timer into bits 40-63 of register T. Bits 0-39 are cleared to zero. The R and S designators are undefined and must be set to zero. This instruction does not deactivate the timer.

When executed in monitor mode, the operation of this instruction is undefined.

3.2.1.57 38 A 64 IN TRANSMIT (R BITS 00-15) TO (T BITS 00-15)

Replace the left-most 16 bits of register T with the left-most 16 bits of register R.

3.2.1.58 39 A 64 NT TRANSMIT REAL-TIME CLOCK TO (T)

Transmit the contents of the real-time clock to bits 16 through 63 of register T. Bits 00 through 15 are cleared. R and S must be zero.

3.2.1.59 3A A 64 NT TRANSMIT (R) TO JOB INTERVAL TIMER

When executed in job mode, this instruction transmits bits 40 through 63 of 64-bit register R to the job interval timer. S and T must be zero. (See Sections 3.1.6.3 and 3.1.8.3).

When executed in monitor mode, this instruction performs as a No op.

3.2.1.60 3B 7 64 BR DATA FLAG REGISTER LOAD/STORE

Transfer the contents of register R to one word of the data flag register and the original contents of that word to register T. The left-most two bits of the S designator specify the word (0-3) of the data flag register. The R and T designators may be the same and this will swap data flag packages. See the Programmer Note in 3.2.1.52 regarding bits which cannot be altered.

NOTE: An immediate data flag branch results at the termination of this instruction if the new contents of the data flag register meet the appropriate conditions.

3.2.1.61 3C 4 32 NT HALF-WORD INDEX MULTIPLY (R)*(S) TO (T)

The right-most 24 bits of registers R and S contain signed, two's complement integers. Their product is formed and stored into the right-most 24 bits of register T. The left-most 8 bits of register T are cleared to zeros.

If the product or either operand exceeds the value, $\pm(2^{23}-1)$ the result is undefined.

3.2.1.62 3D 4 64 NT INDEX MULTIPLY (R)*(S) TO (T)

The right-most 48 bits of registers R and S contain signed, two's complement integers. Their product is formed and stored into the right-most 48 bits of register T. The left-most 16 bits of register T are cleared to zeros.

If the product or either operand exceeds the value, $\pm(2^{47}-1)$ the result is undefined.

3.2.1.63 3E 6 64 IN ENTER (R) WITH I (16 BITS)

Clear register R and transfer the right-most 16 bits of this instruction to the right-most 48 bits of register R (the sign of the 16-bit immediate operand is extended through bit 16).

3.2.1.64 3F 6 64 IN INCREASE (R) BY I (16 BITS)

Replace the right-most 48 bits of register R by the sum of those bits and the right-most 16 bits of this instruction (the sign of the 16-bit immediate operand is extended through bit 16 for the addition). Arithmetic overflow is ignored.

3.2.1.65	40	4	32	RG	ADD U; (R)+(S) TO (T)
3.2.1.66	41	4	32	RG	ADD L; (R)+(S) TO (T)
3.2.1.67	42	4	32	RG	ADD N; (R)+(S) TO (T)
3.2.1.68	43	ILLEGAL			
3.2.1.69	44	4	32	RG	SUB U; (R)-(S) TO (T)
3.2.1.70	45	4	32	RG	SUB L; (R)-(S) TO (T)
3.2.1.71	46	4	32	RG	SUB N; (R)-(S) TO (T)
3.2.1.72	47	ILLEGAL			
3.2.1.73	48	4	32	RG	MPY U; (R)*(S) TO (T)
3.2.1.74	49	4	32	RG	MPY L; (R)*(S) TO (T)
3.2.1.75	4A	ILLEGAL			
3.2.1.76	4B	4	32	RG	MPY S; (R)*(S) TO (T)
3.2.1.77	4C	4	32	RG	DIV U; (R)/(S) TO (T)

These instructions perform the indicated floating-point arithmetic operation on the 32-bit floating-point operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

U signifies that the upper result of the operation is returned; L signifies the lower result; S signifies the significant result; and N signifies the normalized upper result.

Data flags: bits 41, 42, 43, 46, and 58

3.2.1.78 4D 6 32 IN HALF-WORD ENTER R WITH I(16 BITS)

Clear register R and transfer the right-most 16 bits of this instruction to the right-most 24 bits of register R (the sign of the 16-bit immediate operand is extended through bit 8).

3.2.1.79 4E 6 32 IN HALF-WORD INCREASE R BY I (16 BITS)

Replace the right-most 24 bits of register R by the sum of those bits and the right-most 16 bits of this instruction (the sign of the 16-bit immediate operand is extended through bit 8 for the addition). Arithmetic overflow is ignored.

3.2.1.80 4F 4 32 RG DIV S; (R)/(S) TO (T)

This instruction performs a divide significant operation on the 32-bit floating-point operands contained in the registers designated by R and S. The result is stored in the register designated by T.

Data flags: bits 41, 42, 43, 46, and 58

3.2.1.81 50 A 32 RG TRUNCATE; (R) TO (T)

Transmit to destination register T the nearest integer whose magnitude is less than or equal to the 32-bit floating-point operand in origin register R. This integer is represented as an unnormalized 32-bit floating-point number having a positive exponent.

If the exponent of the source operand is positive (greater than or equal to zero), the operand is transmitted directly to the destination register.

If the exponent of the source operand is negative, the magnitude of the coefficient is shifted right end off, and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Zeros are extended on the left during the shift. If the coefficient of the source operand is positive, the shifted coefficient with zero exponent is entered into the destination register. If the coefficient of the source operand is negative, the two's complement of the shifted coefficient with zero exponent is entered into the destination register.

If machine zero is used as an operand, 32 zeros are returned as a result.

Data flags: bits 46 and 58

3.2.1.82 51 A 32 RG FLOOR; (R) TO (T)

Transmit to destination register T the nearest integer less than or equal to the 32-bit floating-point operand in origin register R. This integer is represented as an unnormalized 32-bit floating-point number having a positive exponent.

If the source operand's exponent is positive (greater than or equal to zero), the operand is transmitted directly to the destination register. If the exponent of the source operand is negative, the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The shifted coefficient with zero exponent is entered into the destination register.

If machine zero is used as an operand, 32 zeros are returned as a result.

Data flags: bits 46 and 58

3.2.1.83 52 A 32 RG CEILING; (R) TO (T)

Transmit to destination register T the nearest integer greater than or equal to the 32-bit floating-point operand in origin register R. This integer is represented as an unnormalized 32-bit floating-point number having a positive exponent.

If the source operand's exponent is positive (greater than or equal to zero), the operand is transmitted directly to the destination register. If the exponent of the source operand is negative, the two's complement of the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The two's complement of the shifted coefficient with zero exponent is entered into the destination register.

If machine zero is used as an operand, 32 zeros are returned as a result.

Data flags: bits 46 and 58

3.2.1.84 53 A 32 RG SIGNIFICANT SQUARE ROOT; (R) TO (T)

Transmit to 32-bit register T the square root of a 32-bit floating-point operand in register R.

Data flags: bits 43, 45, 46, and 58

3.2.1.85 54 4 32 RG ADJUST SIGNIFICANCE; (R) PER (S) TO (T)

Adjust the significance of the floating-point operand in register R and transmit it to result register T.

A signed, two's complement, integer is contained in the right-most 24 bits of register S. The absolute value of this integer is a shift count.

If the shift count is positive, shift the operand's coefficient left the number of places specified by the shift count or by the number of shifts needed to normalize the coefficient, whichever is smaller. In either case, the exponent of the operand is reduced by one for each place actually shifted. An all zero coefficient will be shifted left the number of places specified.

If the shift count is negative, shift the operand's coefficient right the number of places specified by the shift count and increase the exponent of the operand by one for each place shifted. If register R is indefinite, register T will be indefinite and data flag bit 46 is set. If register R is machine zero, register T will be machine zero and data flag bit 43 will be set.

This instruction is undefined if the absolute value of the shift count is greater than 23_{10}. Note that the addition of the shift count can cause either exponent overflow or exponent underflow.

Data flags: bits 42, 43, 46, and 58

3.2.1.86 55 4 32 RG ADJUST EXPONENT; (R) PER (S) TO (T)

Transmit the adjusted operand from register R to result register T. The exponent of the result is set equal to the exponent of the operand in register S. The coefficient of the result is formed by shifting the coefficient of the operand from register R.

The shift count used is the difference between the exponents in registers R and S. If the exponent in register R is greater/less than the exponent in register S, the shift is to the left/right, respectively. For zero coefficients in register R, the exponent from register S is copied to register T with an all-zero coefficient.

If a left shift exceeds the number of places required for normalization, the result is set to indefinite, and data flag bit 42 is set. If either or both operands are indefinite or machine zero, the result is set to indefinite. In this case, data flag bit 46 is set and data flag bit 42 is not set.

Data flags: bits 42, 46, and 58

3.2.1.87 56 ILLEGAL

3.2.1.88 57 ILLEGAL

3.2.1.89 58 A 32 RG TRANSMIT; (R) TO (T)

Transmit the operand in 32-bit register R to 32-bit register T.

3.2.1.90 59 A 32 RG ABSOLUTE; (R) TO (T)

Transmit the absolute value of the 32-bit floating-point operand in register R to register T.

Data flags: bits 42, 43, 46, and 58

3.2.1.91 5A A 32 RG EXPONENT OF (R) TO (T)

Transmit the exponent from the left-most 8 bit positions of the origin register R to the right-most 8 bit positions of destination register T. The sign of the exponent is extended through bit 8 of destination register T, the left-most 8 bits of the destination register are cleared to zeros.

3.2.1.92 5B 4 32 RG PACK; (R), (S) TO (T)

Transmit a 32-bit floating-point number to the destination register T. The exponent of the number is obtained from the right-most 8 bit positions of register R and the coefficient is obtained from the right-most 24 bit positions of register S.

3.2.1.93 5C A B RG EXTEND; 32-BIT (R) TO 64-BIT (T)

Extend the floating-point number from 32-bit register R into a 64-bit floating-point number and transmit the result to 64-bit register T. The value of the resulting 16-bit exponent is 24 less than that of the origin operand's exponent. The coefficient is obtained by transmitting the right-most 24 bits of the origin register into bits 16 through 39 of the destination register. The right-most 24 bits of the destination register are cleared to zero.

If R is indefinite, T will be indefinite and data flag bit 46 will be set. If R is machine zero, T will be machine zero and data flag bit 43 will be set

Data flags: bits 43, 46, and 58

3.2.1.94 5D A B RG INDEX EXTEND; 32-BIT (R) TO 64-BIT (T)

Extend the floating-point number from 32-bit register R into a 64-bit floating-point number and transmit the result to 64-bit register T. The value of the resulting 16-bit exponent is the same as the origin operand's exponent. The coefficient is obtained by transmitting the right-most 24 bits of the origin register into bits 40 through 63 of the destination register. Bits 16 through 39 of the destination register are set to the sign of the origin coefficient.

If register R is indefinite, register T will be indefinite and data flag bit 46 will be set. If register R is machine zero, register T will be machine zero and data flag bit 43 will be set.

Data flags: bits 43, 46, and 58

3.2.1.95 5E 7 32 NT LOAD; (T) PER (S), (R)
3.2.1.96 5F 7 32 NT STORE; (T) PER (S), (R)

Load/store 32-bit register T from/into the address specified by (R) + (S) where (R) is the base address and (S) is an item count of half-words. Note that S and R are 64-bit registers and that the item count is shifted left five places before the addition. Overflow from this addition is ignored, if it occurs.

3.2.1.97 60 4 64 RG ADD U; (R)+(S) TO (T)
3.2.1.98 61 4 64 RG ADD L; (R)+(S) TO (T)
3.2.1.99 62 4 64 RG ADD N; (R)+(S) TO (T)

These instructions perform the indicated floating-point arithmetic operation on the 64-bit floating-point operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

U signifies that the upper result of the operation is returned; L signifies the lower result; and N signifies the normalized upper result.

Data flags: bits 42, 43, 46, and 58

3.2.1.100 63 4 64 RG ADD ADDRESS; (R)+(S) TO (T)

This instruction adds bits 16 through 63 of register R to bits 16 through 63 of register S and stores the result in bits 16 through 63 of register T. Bits 16 through 63 are treated as 48-bit, positive, unsigned integers. Arithmetic overflow is ignored. Bits 0 through 15 of register R are transferred without modification to bits 0 through 15 of register T.

3.2.1.101 64 4 64 RG SUB U; (R)-(S) TO (T)
3.2.1.102 65 4 64 RG SUB L; (R)-(S) TO (T)
3.2.1.103 66 4 64 RG SUB N; (R)-(S) TO (T)

These instructions perform the indicated floating-point arithmetic operation on the 64-bit floating-point operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

U signifies that the upper result of the operation is returned; L signifies the lower result; and N signifies the normalized upper result.

Data flags: bits 42, 43, 46, and 58

3.2.1.104 67 4 64 RG SUB ADDRESS; (R)-(S) TO (T)

This instruction subtracts bits 16 through 63 of register S from bits 16 through 63 of register R and stores the result in bits 16 through 63 of register T. Bits 16 through 63 are treated as 48-bit, positive unsigned integers. Arithmetic overflow is ignored. Bits 0 through 15 of register R are transferred without modification to bits 0 through 15 of register T.

3.2.1.105	68	4	64	RG	MPY U; (R)*(S) TO (T)
3.2.1.106	69	4	64	RG	MPY L; (R)*(S) TO (T)
3.2.1.107	6A	ILLEGAL			
3.2.1.108	6B	4	64	RG	MPY S; (R)*(S) TO (T)
3.2.1.109	6C	4	64	RG	DIV U; (R)/(S) TO (T)

These instructions perform the indicated floating-point arithmetic operation on the 64-bit floating-point operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

U signifies that the upper result of the operation is returned; L signifies the lower result; S signifies the significant result.

Data flags: bits 41, 42, 43, 46, and 58

3.2.1.110 6D 4 64 RG INSERT BITS; (R) TO (T) PER (S)

This instruction inserts the right-most bits of the register designated by R into the register designated by T.

```
         ------------------------------------------------------
         |                                        |    m     |
Reg R    |                                        |<-------->|
         |                                        |   bits   |
         ------------------------------------------------------
                                                       |
                                   INSERT              v
                       ---------------------------------
                       |
                       v
         ------------------------------------------------------
         |      *     | |   m     |          *               |
Reg T    |            |<-------->|                           |
         |            | |  bits   |                          |
         ------------------------------------------------------
                      ^       * These bits are unaltered
                      |
                  bit n

         ------------------------------------------------------
         |             |         |                    |      |
Reg S    | 0---0       |    m    | 0----------------0 |  n   |
         |             |         |                    |      |
         ------------------------------------------------------
         0          9 10     15 16                 57 58   63
```

Bits 10 through 15 of register S contain the number (m) of right-most bits to be inserted. The right-most 6 bits of register S specify the the bit number (n) in register T where the leftmost bit of the inserted data will be placed. Bits 0 through 9 and 16 through 57 of register S are undefined and must be set to zero.

If the R designator is equal to zero, then register zero will provide machine zero. If m plus n is greater than 64_{10}, or if m is equal to zero, the results of this instruction are undefined.

3.2.1.111 6E 4 64 RG EXTRACT BITS; (R) TO (T) PER (S)

This instruction extracts bits from register R and stores them into the right-most portion of register T. Register T is cleared before receiving the extracted bits.

```
         ----------------------------------------------------------
         |          | |  m     |                                  |
Reg R    |          |<-------->|                                  |
         |          | | bits   |                                  |
         ----------------------------------------------------------
                     ^   |
                     |   v                EXTRACT
                  bit n  ----------------------------------
                                                           |
                                                           v
         ----------------------------------------------------------
         |                                          |     m      |
Reg T    | 0--------------------------------------0 |<---------->|
         |                                          |    bits    |
         ----------------------------------------------------------

         ----------------------------------------------------------
         |        |        |                          |          |
Reg S    | 0---0  |   m    | 0----------------------0 |    n     |
         |        |        |                          |          |
         ----------------------------------------------------------
         0      9 10    15 16                      57 58       63
```

Bits 10 through 15 of register S contain the number (m) of bits to be extracted from register R. The right-most 6 bits of register S specify the left-most bit number of the extracted bits. Bits 0 through 9 and 16 through 57 of register S are undefined and must be set to zero.

If the R designator is equal to zero, register zero will provide machine zero. If m plus n is greater than 64_{10}, or if m is equal to zero, the results of this instruction are undefined.

3.2.1.112 6F 4 64 RG DIV S; (R)/(S) TO (T)

This instruction performs a Divide Significant operation on the 64-bit floating-point operands contained in the registers designated by R and S. The result is stored in the register designated by T.

Data flags: bits 41, 42, 43, 46, and 58

3.2.1.113 70 A 64 RG TRUNCATE; (R) TO (T)

Transmit to destination register T the nearest integer whose magnitude is less than or equal to the magnitude of the 64-bit floating-point operand in origin register R. The integer is represented as an unnormalized 64-bit floating-point number having a positive exponent.

If the exponent of the source operand is positive (greater than or equal to zero), the operand is transmitted directly to the destination register.

If the exponent of the source operand is negative, the magnitude of the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Zeros are extended on the left during the shift. If the coefficient of the source operand is positive, the shifted coefficient with zero exponent is entered into the destination register. If the coefficient of the source operand is negative, the two's complement of the shifted coefficient with zero exponent is entered into the destination register.

If a machine zero is used as an operand, 64 zeros are returned as a result.

Data flags: bits 46 and 58

3.2.1.114 71 A 64 RG FLOOR; (R) TO (T)

Transmit to destination register T the nearest integer less than or equal to the 64-bit floating-point operand in origin register R. This integer is represented as an unnormalized 64-bit floating-point number having a positive exponent.

If the source operand's exponent is positive (greater than or equal to zero), the operand is transmitted directly to the destination register. If the exponent of the source operand is negative, the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The shifted coefficient with zero exponent is entered into the destination register.

If a machine zero is used as an operand, 64 zeros are returned as a result.

Data flags: bits 46 and 58

3.2.1.115 72 A 64 RG CEILING; (R) TO (T)

Transmit to destination register T the nearest integer greater than or equal to the 64-bit floating-point operand in origin register R. This integer is represented as an unnormalized 64-bit floating-point number having a positive exponent.

If the source operand's exponent is positive (greater than or equal to zero), the operand is transmitted directly to the destination register. If the exponent of the source operand is negative, the two's complement of the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The two's complement of the shifted coefficient with zero exponent is entered into the destination register.

If machine zero is used as an operand, 64 zeros are returned as a result.

Data flags: bits 46 and 58

3.2.1.116 73 A 64 RG SIGNIFICANT SQUARE ROOT; (R) TO (T)

Transmit to register T the square root of the 64-bit floating-point operand in register R.

Data flags: bits 43, 45, 46, and 58

3.2.1.117 74 4 64 RG ADJUST SIGNIFICANCE; (R) PER (S) TO (T)

Adjust the significance of the floating-point operand in register R and transmit it to result register T.

A signed, two's complement integer is contained in the right-most 48 bits of register S. The absolute value of this integer is a shift count. The left-most 16 bits of register S are ignored.

If the shift count is positive, shift the operand's coefficient left the number of places specified by the shift count or by the number of shifts needed to normalize the coefficient, whichever is smaller. In either case, the exponent of the operand is reduced by one for each place actually shifted. An all zero coefficient will be shifted left the number of places specified.

If the shift count is negative, shift the operand's coefficient right the number of places specified by the shift count and increase the exponent of the operand by one for each place shifted.

This instruction is undefined if the absolute value of the shift count is greater than 47_{10}. Note that the addition of shift count can cause either exponent overflow or exponent underflow.

If register R is indefinite, register T will be indefinite and data flag bit 46 will be set. If register R is machine zero, register T will be machine zero and data flag bit 43 will be set.

Data flags: bits 42, 43, 46, and 58

3.2.1.118 75 4 64 RG ADJUST EXPONENT; (R) PER (S) TO (T)

Transmit the adjusted operand from register R to result register T. The exponent of the result is set equal to the exponent of the operand in register S. The result is formed by shifting the coefficient of the operand from register R.

The shift count used is the difference between the exponents is register R and S. If the exponent in register R is greater/less than the exponent in register S, the shift is to the left/right, respectively. For zero coefficients in register R, the exponent from register S is copied to register T with an all-zero coefficient.

If a left shift exceeds the number of places required for normalization, the result is set to indefinite and data flag 42 is set. If either or both operands are indefinite or machine zero, the result is set to indefinite. In this case, data flag bit 46 is set and data flag bit 42 is not set.

Data flags: bits 42, 46, and 58

3.2.1.119 76 A B RG CONTRACT; 64-BIT (R) TO 32-BIT (T)

Contract the 64-bit floating-point number from register R into a 32-bit floating-point number and transmit the result to 32-bit register T.

Input Exponent	Result
7FFF : 7000	Result Indefinite Indefinite Data Flag 46
6FFF : 0058	Result Indefinite Data Flag 42, 46
0057 : : : : FF78	Result exponent 24_{10} larger than input exponent Copy left-most 24 bits of input coefficient
FF77 : 8000	Result machine zero Data Flag 43

The 24-bit result coefficient is copied from the left-most 24 bits of the 48-bit source coefficient (bits 16 through 39). This has the effect of contracting all negative source coefficients, whose absolute values (neglecting the exponent) were less than or equal to 2^{24}, to a minus one.

Data flags: bits 42, 43, 46, and 58

3.2.1.120 77 A B RG ROUNDED CONTRACT; 64-BIT (R) TO 32-BIT (T)

Perform a rounded contract operation on the 64-bit floating-point number in register R and transmit the 32-bit floating-point result to 32-bit register T. A positive one is added to the origin operand in bit position 40. If overflow occurs the exponent is increased by one and the coefficient is shifted right one place. The left-most 24 bits of this 48-bit sum are then transmitted to the 24-bit coefficient portion of register T. Each non-endcase result 8-bit exponent is 24_{10} (25_{10} if overflow occurred) greater than the corresponding source exponent.

Data flags: bits 42, 43, 46, and 58

3.2.1.121 78 A 64 RG TRANSMIT; (R) TO (T)

Transmit the 64-bit operand in register R to register T.

3.2.1.122 79 A 64 RG ABSOLUTE; (R) TO (T)

Transmit the absolute value of the 64-bit floating-point operand in register R to register T.

Data flags: bits 42, 43, 46, and 58

3.2.1.123 7A A 64 RG EXPONENT OF (R) TO (T)

Transmit the exponent from the left-most 16 bit positions of origin register R to the right-most 16 bit positions of destination register T. The sign of the exponent is extended through bit 16 of destination register T. The left-most 16 bits of the destination register are cleared to zeros.

3.2.1.124 7B 4 64 RG PACK; (R), (S) TO (T)

Transmit a 64-bit floating-point number to destination register T. The exponent of the number is obtained from the right-most 16 bit positions of register R, and the coefficient is obtained from the right-most 48 bit positions of register S.

3.2.1.125 7C A 64 RG LENGTH; (R) TO (T)

Transmit the left-most 16 bit positions of origin register R to the right-most 16 bit positions of destination register T. The left-most 48 bits of the destination register are cleared to zeros.

3.2.1.126 7D 7 64 NT SWAP; S----->T AND R----->S

Move to destination field T, a portion of the Register File beginning at the 64-bit register specified by the right-most eight bits of register S. Transmit source field R to the Register File beginning at the 64-bit register specified by the right-most eight bits of register S.

The left-most 16 bits of register R and T specify the field length in words for the source and destination fields, respectively. The field lengths of the source and destination fields may be different but each must be even. A zero field length indicates no transfer for that field. Any transfer of words into or out of the Register File that becomes exhausted of registers (i.e., beyond the bounds of the Register File), causes the instruction to become undefined.

The right-most 48 bits of registers R and T specify the base address of the source and destination fields, respectively. These addresses must specify an even 64-bit word in Main Memory. Bits 57 through 63 of register R and T are undefined and must be set to zero. Overlap of the source and destination fields is allowed only if the base addresses for both fields are equal.

Registers R, S, or T, may be in the range of the registers being swapped.

The starting register in the file specified by the right-most eight bits of register S must be an even register or this instruction will be treated as an undefined instruction. For additional material see Section 3.1.7 on the Register File.

3.2.1.127 7E 7 64 NT LOAD; (T) PER (S), (R)
3.2.1.128 7F 7 64 NT STORE; (T) PER (S), (R)

Load/store 64-bit register T from/into the address specified by (R) + (S) where (R) is the base address and (S) is an item count of words.

3.2.1.129 80 ILLEGAL

3.2.1.130 81 ILLEGAL

3.2.1.131 82 ILLEGAL

3.2.1.132 83 ILLEGAL

3.2.1.133 84 ILLEGAL

3.2.1.134 85 ILLEGAL

3.2.1.135 86 ILLEGAL

3.2.1.136 87 ILLEGAL

3.2.1.137 88 ILLEGAL

3.2.1.138 89 ILLEGAL

3.2.1.139 8A ILLEGAL

3.2.1.140 8B ILLEGAL

3.2.1.141 8C ILLEGAL

3.2.1.142 8D ILLEGAL

3.2.1.143 8E ILLEGAL

3.2.1.144 8F ILLEGAL

3.2.1.145 90 ILLEGAL

3.2.1.146 91 ILLEGAL

3.2.1.147 92 ILLEGAL

3.2.1.148 93 ILLEGAL

3.2.1.149 94 ILLEGAL

3.2.1.150 95 ILLEGAL

3.2.1.151 96 ILLEGAL

3.2.1.152 97 ILLEGAL

3.2.1.153 98 ILLEGAL

3.2.1.154 99 ILLEGAL

3.2.1.155 9A ILLEGAL

3.2.1.156 9B ILLEGAL

3.2.1.157 9C ILLEGAL

3.2.1.158 9D D E SM MEMORY MAP

This instruction controls memory to memory data transfers with reordering capability; the transfers can be intramemory for either the Main Memory or the Intermediate Memory, or they can be intermemory transfers for certain cases from Main Memory to Intermediate Memory or vice versa. The operations provided by this instruction permit remapping of data in either memory while other vector operations are in process in the Vector Unit. In addition, mapping operations can be executing in both Main Memory and Intermediate Memory concurrently if no resources are common to both. Setup information is provided to three main memory read ports, READ1, READ2, and READ3 and to one main memory write port, WRITE1, and/or to three intermediate memory read/write ports, RW1, RW2, and RW3. Referring to instruction format D, the fields are defined as follows:

Field	Function and Meaning
F	Function Code - 8 bits (9D) - Memory Map
K	Read Key - 5 bits
Z	Write Key - 5 bits
S	Suboperation - 3 bits - these codes identify the suboperation to be performed; refer to Suboperation Descriptions, below.

Code	Meaning
0	No Operation
1	Gather
2	Scatter
3	Illegal
4	Illegal
5	Compress
6	Mask
7	Merge

(continued)

3.2.1.158 (Cont.)

Field	Function and Meaning
	Source/Destination - 2 bits - specifies Main Memory and/or Intermediate Memory as source and/or destination for operand streams according to the following codes.

Code	Meaning
0	Main Memory for both source and destination.
1	Main Memory as source and Intermediate Memory as destination -- scatter suboperation only (S=2), results are undefined for S-codes 1,5,6,7.
2	Intermediate Memory as source and Main Memory as destination -- gather and compress suboperations only (S=1,5), results are undefined for S-codes 2,6,7.
3	Intermediate Memory as both source and destination.

B Operand Size - 1 bit - specifies size of operands for memory to memory data transfer.

Code	Meaning
0	64-bit operands.
1	32-bit operands.

C,D,E Suboperation Modifiers - 2, 1, and 1 bits, respectively - these modifiers have different meaning dependent on the suboperation, and are defined below with each suboperation description.

(continued)

3.2.1.158 (Cont.)

Field	Function and Meaning
T,U, V,W	Register Designators - 8 bits - each 8-bit field holds a register file designator; the word specified by T, U, V, or W will be sent to the appropriate read and write ports in the two map units for each of the T, U, V, or W designators which is non-zero. If a designator is zero, the respective read or write bus control in either map unit is not used by the instructon. Generally, the information in the word specified by T, U, V, or W consists of a length (number of operands) and a memory address for the respective data stream.
X,Y	Null Fields - 1 and 3 bits, respectively - not used, must be zero.

Suboperation Descriptions

Gather -- S field = 1

The gather suboperation uses a list of memory locations to gather data into another list. The list of memory locations can be either explicit or implicit. If explicit, an index list is read from memory and each index is added to a base address to fetch the operand list. An implicit list is specified by declaring a stride in the instruction (see C field description below). The base address for the index list, if explicit, and the operand list are contained in bits 16-63 of the registers designated by V and T, respectively.

The base address of the destination field is contained in bits 16-63 of the register designated by W, and the total number of records to be transferred is the field length contained in bits 00-15 of the register designated by W. If a

(continued)

3.2.1.158 (Cont.)

Suboperation Descriptions

Gather (continued)

2-dimension Gather is specified, the stride in the Y-direction (STRIDEY) is contained in bits 16-63 of the register designated by V. If a 3-dimension Gather is specified, the stride in the Z-direction STRIDEZ) is contained in bits 16-63 of the register designated by U; STRIDEY is provided as for the 2-dimension stride and, in addition, bits 00-15 of the register designated by V contain YCOUNT, the number of times STRIDEY is applied before STRIDEZ is applied.

Length of records is specified by the length field (bits 00-15) of the register designated by T; e.g., a field length of one transfers one word for each address in the list, or stride, while a field length of 100 transfers 100 successive words for each address in the list, or stride, starting at that address. Note that a field length of zero results in no data being transferred.

For the gather suboperation, the C, D, and E fields are defined as follows:

C - Stride Specification - 2 bits

0 An index list (no stride) is used from the same memory as the source operands.

1 An index list (no stride) is used from Intermediate Memory if the source is Main Memory for operands, and vice versa.

2 2-dimension stride.

3 3-dimension stride.

(continued)

3.2.1.158 (Cont.)

Suboperation Descriptions

Gather (continued)

D - STRIDEY Indirect - 1 bit

If C=0 or C=1, D is ignored and the register designated by V contains a memory address; if C=2 or C=3, D is as follows:

0 Register designated by V contains STRIDEY.

1 Register designated by V contains a memory address, the content of which is STRIDEY.

E - STRIDEZ Indirect - 1 bit

If the C field is not equal to 3, E is ignored; for C equal to 3, E is as follows:

0 Register designated by U contains STRIDEZ.

1 Register designated by U contains a memory address, the content of which is STRIDEZ.

Scatter -- S field = 2

The scatter suboperation uses a list of memory locations to scatter a list of input data back into memory - the inverse of Gather. The destination list of memory locations can be either explicit or implicit. If explicit, an index list is read from memory and each index is added to the destination (write) base address to provide addresses for the write operand list. An implicit list is specified by declaring a stride in the instruction (see C field description below). The base address for the source operand list and the index list are contained in bits 16-63 of the register designated by T and V, respectively.

The base address of the destination list is contained in bits 16-63 of the register designated by W, and the total number of records to be transferred is the field length contained in bits

(continued)

3.2.1.158 (Cont.)

Suboperation Descriptions

Scatter (continued)

00-15 of the register designated by W. If a 2-dimension Scatter is specified, the stride in the Y-direction (STRIDEY) is contained in bits 16-63 of the register designated by V. If a 3-dimension Scatter is specified, the stride in the Z-direction (STRIDEZ) is contained in bits 16-63 of the register designated by U; STRIDEY is provided as for the 2-dimension stride and, in addition, bits 00-15 of the register designated by V contain YCOUNT, the number of times STRIDEY is applied before STRIDEZ is applied.

Length of records is specified by the length field (bits 00-15) of the register designated by T; e.g., a field length of one transfers one word for each address in the list, or stride, while a field length of 100 transfers 100 successive words for each address in the list, or stride, starting at that address. Note that a field length of zero results in no data being transferred.

For the scatter suboperation, the C, D, and E fields are defined as follows:

C - Stride Specification - 2 bits

0 An index list (no stride) is used from the same memory as the source operands.

1 An index list (no stride) is used from Intermediate Memory if the source is Main Memory for operands, and vice versa.

2 2-dimension stride.

3 3-dimension stride.

(continued)

3.2.1.158 (Cont.)

Suboperation Descriptions

Scatter (continued)

D - STRIDEY Indirect - 1 bit

If C=0 or C=1, D is ignored and the register designated by V contains a memory address; if C2 or C=3, D is as follows:

0 Register designated by V contains STRIDEY.

1 Register designated by V contains a memory address, the content of which is STRIDEY.

E - STRIDEZ Indirect - 1 bit

If the C field is not equal to 3, E is ignored; for C equal to 3, E is as follows:

0 Register designated by U contains STRIDEZ.

1 Register designated by U contains a memory address, the content of which is STRIDEZ.

Compress -- S field = 5

The compress suboperation deletes operands from a source operand stream according to a control vector (bit string), writing result data to a destination operand stream.

The register designated by T contains the source operand base address in bits 16-63. Bits 00-15 and 16-63 of the register designated by V contain the vector length and control vector base address, respectively. The register designated by W contains the write vector length and destination base address in bits 00-15 and 16-63, respectively.

For the compress suboperation, the C, D, and E fields are defined as follows:

(continued)

3.2.1.158 (Cont.)

Suboperation Descriptions

Compress (continued)

C - Control Vector Operation - 2 bits

0 Operands corresponding to each zero bit in the control vector are deleted from the result stream.

1 Operands corresponding to each one bit in the control vector are deleted from the result stream.

2 Illegal.

3 Illegal.

D - Repeat Control Vector - 1 bit

0 Control vector extension: if the control vector length becomes exhausted before satisfying the write vector length, the control vector (bit string) is extended with permissive elements (1's for C=0, 0's for C=1) until the write vector length is satisfied.

1 Control vector repeat: if the control vector length becomes exhausted before satisfying the write vector length, the control vector (bit string) is repeated from the beginning until the write vector length is satisfied.

E - Control Vector Source - 1 bit - If the B field does not equal 2, this field is ignored; if the B field does equal 2 (Intermediate Memory to Main Memory), this field specifies the source of the control vector (bit string) as follows:

0 Control vector source is Intermediate Memory.

1 Control vector source is Main Memory.

(continued)

3.2.1.158 (Cont.)

Suboperation Descriptions

Mask/Merge (continued)

Mask/Merge -- S field = 6 or 7, respectively

Suboperations Mask and Merge both combine two source operand streams according to a control vector (bit string), into one destination (write) operand stream. In a Mask (S=6) the operands of the stream not being selected are skipped (the ith element of A or the ith element of B to the result). In a Merge (S=7) no operands are skipped so that the "top" operand of each stream is always available to be taken (the mth element of A or the nth element of B to the result). Merge thus corresponds to shuffling cards and Mask corresponds to repeatedly selecting one of two from a pair of cards. Note, however, that the E field allows a combined mask/merge suboperation (one source stream is skipped, the other is not).

The base addresses for the A and B streams are contained in bits 16-63 of the registers designated by T and U, respectively. The control vector length and base address are contained in bits 00-15 and 16-63, respectively, of the register designated by V. The register designated by W contains the write vector length and destination base address in bits 00-15 and 16-63, respectively.

For the mask/merge suboperations the C, D, and E fields are defined as follows:

3.2.1.158 (Cont.)

Suboperation Descriptions

Mask/Merge (continued)

C - Control Vector Operation - 2 bits

0 The A operand is sent to the write stream for each "one" bit in the control vector; the B operand is sent for each "zero" bit in the control vector.

1 The A operand is sent to the write stream for each "zero" bit in the control vector; the B operand is sent for each "one" bit in the control vector.

2 Illegal.

3 Illegal.

D - Repeat Control Vector - 1 bit

0 Control vector extension: if the control vector length becomes exhausted before satisfying the write vector length, the control vector (bit string) is extended with permissive elements (1's for C=0, 0's for C=1) until the write vector length is satisfied.

1 Control vector repeat: if the control vector length becomes exhausted before satisfying the write vector length, the control vector (bit string) is repeated from the beginning until the write vector length is satisfied.

3.2.1.158 (Cont.)

Suboperation Descriptions

Mask/Merge (continued)

E - Decompress - 1 bit

This bit is ignored for a mask suboperation (S=6); for the merge suboperation (S=7) the E field has the following meaning:

0 Merge B operands: the B stream operand is not skipped if an operand is taken from the A steam.

1 Decompress: one operand from the B stream is skipped for each operand taken from the A stream.

This bit does not effect the A stream which is never skipped on a merge suboperation; it thus allows a combination mask/merge suboperation.

3.2.1.159 9E E E SM VECTOR READ PORT SETUP

This instruction provides setup for vector read stream control between the Vector Unit and Main Memory. A total of four read control elements may receive setup information through execution of this instruction; the control elements are four memory read buses, VR1, VR2, VR3, and VR4.

Referring to instruction format E, the fields and their codes are defined below.

Field	Function and Meaning
F	Function Code - 8 bits (9E) - Vector Read Port Setup
K	Read Key - 5 bits
A,B, C,D	Operand Size - 1 bit - for input trunks A, B, C, and D, respectively.

Code	Meaning
0	64-bit oerands.
1	32-bit operands.

(continued)

E,G, H,J — Extend/Repeat Vector - 2 bits - for vector trunks A, B, C, and D, respectively. This field specifies the operand for fill when a source vector ends before operation completes:

Code	Meaning
0	Extend indefinite.
1	Extend floating-point zero.
2	Extend floating-point one.
3	Repeat input vector.

When an input vector is repeated (code 3), the input port restarts the vector from the beginning after the number of input operands specified by the field length

(continued)

3.2.1.159 (Cont.)

Field	Function and Meaning
	have been transferred. A length of one with repeat means that the single operand is held for the full execution time of the instruction. A length of zero with repeat is a special case that causes an operand to be replicated eight times before proceeding to the next operand.
M	VR2 Setup - 1 bit - specifies type of setup for read port VR2; read ports VR1, VR3, and VR4 can be set up only for streaming mode, but VR2 can be set up for streaming operands or for control vector addressing and/or fetch. Code / Meaning 0 Set up VR2 in streaming mode. 1 Set up VR2 in control vector mode.
T,U, V,W	Register File Designator - 8 bits - codes to designate one of 256 registers to supply setup information such as address and field length; T, U, V, and W apply to read ports VR1, VR2, VR3, and VR4, respectively, in the Vector Streaming Unit; these ports supply source streams SR1, SR2, SR3, and SR4, respectively, to the Vector Units. If field T, U, V, or W is zero, the respective port is not used by the instruction.
L,N,P	Null Fields - 1 bit - not used, must be zero.
X	Null Field - 3 bits - not used, must be zero.

3.2.1.160 9F F E SM VECTOR ARITHMETIC

This instruction specifies the arithmetic operations, selects input data trunks, and provides setup streams) for the Vector Unit. Memory ports supplying source operand streams from memory to the input data trunks must be set up with the 9E instruction prior to execution of this instruction.

Referring to instruction format F, the fields are defined as follows:

Field	Function and Meaning
F	Function Code - 8 bits (9F) - Vector Arithmetic
Z	Write Key - 5 bits
S	Suboperation - 6 bits - This code identifies the suboperation to be performed; refer to Suboperation Descriptions, below.
A,B, C,D	Source - 2 bits - specifies source for A, B, C, and D input streams, respectively, as follows:

Code	Source
0	SR1 from Vector Streaming Unit.
1	SR2 from Vector Streaming Unit.
2	SR3 from Vector Streaming Unit.
3	SR4 from Vector Streaming Unit.

G,H B and D Operand Sign Control - 2 bits - select sign and magnitude of the B and D operand streams, respectively, according to the following codes; A and C operand streams cannot be modified.

Code	Meaning
0	Operands unchanged.
1	Complement operands.
2	Positive magnitude of operands.
3	Negative magnitude of operands.

(continued)

3.2.1.160 (Cont.)

Field	Function and Meaning
L,M	Operand Size - 1 bit - for result streams AW1 and AW2, respectively.

Code	Meaning
0	64-bit operands.
1	32-bit operands.

Field	Function and Meaning
N,P	Round Result - 1 bit - round specification for result streams AW1 and AW2, respectively; results are rounded or not rounded according to the following code:

Code	Meaning
0	Do not round result.
1	Round result.

Field	Function and Meaning
Q,R	Control Vector - 4 bits - these fields apply to ouputs AW1 and AW2, respectively, to specify generation of a control vector or use of a control vector during the write to Main Memory.

Code	Meaning
0-7	No control vector.
8	Use normal control vector.
9	Generate control vector with test =.
A	Generate control vector with test <.
B	Generate control vector with test $\leq$.
C	Use inverted control vector.
D	Generate control vector with test $\neq$.
E	Generate control vector with test $\geq$.
F	Generate control vector with tst >.

(continued)

3.2.1.160 (Cont.)

Field	Function and Meaning
	VR2 in the Vector Streaming Unit is the only source of a control vector and must be set up with a 9E instruction prior to specifying use of a control vector in this instruction.
X,Y	Test Stream - 1 bit - this code specifies which result stream, AW1 or AW2, is to be tested for generation of a control vector on write ports VW1 and VW2, respectively, in the Vector Streaming Unit. X and Y apply only for the Q and R fields, respectively, equal to 9 through B and D through F. Code / Meaning 0 Test AW1. 1 Test AW2.
V,W	Register File Designator - 8 bits - codes 00 through FF (hex) designate one of 256 registers to supply setup information such as address and field length; V and W apply to write ports VW1 and VW2, respectively, in the Vector Streaming Unit for writing result streams AW1 and AW2, respectively, to Main Memory. If field V or W is zero, the respective write control in the Vector Streaming Unit is not used.
E,T,U	Null Fields - 1 bit each - not used, must be zero.

(continued)

3.2.1.160 (Cont.)

Suboperation Descriptions

The 6-bit suboperation code, S, defines the arithmetic operatione to be performed by the Vector Unit producing results on two buses, AW1 and AW2. The codes and their meanings are defined below. Suboperations 02-14, 21, 23, 25, and 30-36 are performed with normalized arithmetic.

Code	Result on AW1 Bus	Result on AW2 Bus
00	A	C
01	B	D
02	A+D	B+C
03	A+D	B*C
04	A*D	B*C
05	(A+B)*D	A+B
06	(A+B)*(C*D)	C*D
07	(A+B)*(C+D)	A+B
08	A*C+D	B
09	(A+B)+D	A+B
0A	(A+B)+(C+D)	C+D
0B	(A+B)+C*D	C*D
0C	(A+B)*C+D	(A+B)*C
0D	(A*B)+(C*D)	C*D
0E	A*(B+C*D)	B+(C*D)
0F	(A+B)*D	(A+B)*C

(continued)

3.2.1.160 (Cont.)

Suboperation Descriptions

Code	Result on AW1 Bus	Result on AW2 Bus
10	(A*C)+D	(A*C)+B
11	(A*B)+D	C*D
12	(A*B)+D	C+D
13	A*(B+(C+D))	B+(C+D)
14	A*B*C+D	B*C
15	Code not used	
16	A+D Upper Sum	B+C Upper Sum
17	A+D Lower Sum	B+C Lower Sum
18	A+D Upper Sum	B+C Lower Sum
19	A+D Upper Sum	B*C Upper Product
1A	A+D Lower Sum	B*C Lower Product
1B	A*D Upper Product	B*C Upper Product
1C	A*D Lower Product	B*C Lower Product
1D	A*D Upper Product	B*C Lower Product
1E	Code not used	
1F	Expand 32-bit B to 64 bits	Expand 32-bit D to 64 bits

(continued)

3.2.1.160 (Cont.)

Suboperation Descriptions

Code	Result on AW1 Bus	Result on AW2 Bus
20	A/C Divide Upper 32-bit mode	None
21	A/C Divide Normalize 32-bit mode	None
22	A/C Divide Upper 64-bit mode (23-bit accuracy)	I
23	A/C Divide Normalize 64-bit mode (23-bit accuracy)	None
24	A*B*D Divide Upper 64-bit mode (47-bit accuracy) See note 1	None
25	A*B*D Divide Normalize 64-bit mode (47-bit accuracy) See note 1	None
26	Code not used	
2F	Code not used	

Note 1 -- A must be the AW1 result and B must be the AW2 result of suboperation 22; D is the divisor of the suboperation 22. B*D is 2's complemented before multiplying by A.

(continued)

3.2.1.160 (Cont.)

Suboperation Descriptions

Code	Result on AW1 Bus	Result on AW2 Bus
30	Sum One $(A_0+A_1+...+A_n)$	None
31	Sum Two $(A_0+B_0)+(A_1+B_1)+...+(A_n+B_n)$	None
32	Sum of Products $(A_0*B_0)+(A_1*B_1)+...+(A_n*B_n)$	None
33	Product $(A_0*A_1*...*A_n)$	None
34	Product of Sums $(A_0+B_0)*(A_1+B_1)*...*(A_n+B_n)$	None
35	Maximum	Count (note 2)
36	Minimum	Count (note 2)
37	Code not used	
¦	¦ ¦ ¦	
3F	Code not used	

Note 2 -- Count is the element number of the operand that corresponds to the maximum or the minimum.

Data flags: bits 41, 42, 43, and 46.

3.2.1.161 A0 ILLEGAL

3.2.1.162 A1 ILLEGAL

3.2.1.163 A2 ILLEGAL

3.2.1.164 A3 ILLEGAL

3.2.1.165 A4 ILLEGAL

3.2.1.166 A5 ILLEGAL

3.2.1.167 A6 ILLEGAL

3.2.1.168 A7 ILLEGAL

3.2.1.169 A8 ILLEGAL

3.2.1.170 A9 ILLEGAL

3.2.1.171 AA ILLEGAL

3.2.1.172 AB ILLEGAL

3.2.1.173 AC ILLEGAL

3.2.1.174 AD ILLEGAL

3.2.1.175 AE ILLEGAL

3.2.1.176 AF ILLEGAL

```
3.2.1.177    B0  C  E   BR   COMPARE, EQUAL
3.2.1.178    B1  C  E   BR   COMPARE, NOT EQUAL
3.2.1.179    B2  C  E   BR   COMPARE, GREATER THAN OR EQUAL
3.2.1.180    B3  C  E   BR   COMPARE, LESS THAN
3.2.1.181    B4  C  E   BR   COMPARE, LESS THAN OR EQUAL
3.2.1.182    B5  C  E   BR   COMPARE, GREATER THAN
```

These instructions perform integer or floating-point compares to establish criteria on which to branch or set conditions dependent on G-bit specifications.

```
B0   C   E    BR  COMPARE INTEGER, BRANCH IF (A) + (X) EQ (Z)
B1   C   E    BR  COMPARE INTEGER, BRANCH IF (A) + (X) NE (Z)
B1   C   E    BR  COMPARE INTEGER, BRANCH IF (A) + (X) GE (Z)
B1   C   E    BR  COMPARE INTEGER, BRANCH IF (A) + (X) LT (Z)
B1   C   E    BR  COMPARE INTEGER, BRANCH IF (A) + (X) LE (Z)
B1   C   E    BR  COMPARE INTEGER, BRANCH IF (A) + (X) GT (Z)
```

If bit 0 of the G designator is cleared/set, registers A, X, C, and Z are 64/32 bits, respectively. Registers B and Y are always 64 bits.

G bits 1 and 2 must be set to zero.

These instructions are executed in the following 5 steps:

1. Form the sum of the 48-bit (24-bit if G bit 0 = 1) integers from the right-most portion of registers A and X, ignoring overflows. If designators A and/or X equal zero, machine zero will be supplied.

2. Read register Z. If the Z designator is equal to zero compare against 48 zeros (24 zeros if G bit 0 = 1) may be made.

3. Store the following in register C:

   - The sum from step 1 is stored into the right-most 48 bits (24 bits if G bit 0 = 1) of register C.

   - The left-most 16 bits (8 bits if G bit 0 = 1) of register A are copied into the left-most portion of register C.

4. Compare the sum formed in step 1 with register Z as follows:

   - G bit 3 = 0 Integers compared are the 48-bit (24 bits if G bit 0 = 1) result of step 1 and the right-most 48 bits (24 bits if G bit 0 = 1) read from register Z in step 2.

(continued)

3.2.1.182 (Cont.)

- G bit 3 = 1 Integers compared are the 64 bits that are stored into register C in step 3 and 64 bits read from register Z in step 2.

  This compare is defined only for the B0 and B1 instructions (EQ and NE).

  When both G bit 0 and G bit 3 are 1 the instructions are undefined.

- G bit 4 = 0 Integers compared are interpreted as signed two's complement numbers.

- G bit 4 = 1 Integers compared are interpreted as unsigned numbers.

The following table indicates the ordering of numbers from largest to smallest as controlled by G bit 4.

	0	1
Largest	7F -------- FF	FF -------- FF
	7F -------- FE	FF -------- FE
	.	.
	.	.
	.	.
	00 -------- 01	80 -------- 01
	00 -------- 00	80 -------- 00
	FF -------- FF	7F -------- FF
	.	.
	.	.
v	.	.
Smallest	80 -------- 01	00 -------- 01
	80 -------- 00	00 -------- 00

(continued)

3.2.1.182 (Cont.)

5. If the specified compare condition is met the instruction performs as follows:

   - G bit 5 = 0 Branch to the address formed by adding the half-word item count from register Y left-shifted 5 places to the base address from register B.

   - G bit 5 = 1 Branch to the address formed by adding (G bit 6 = 0) or subtracting (G bit 6 = 1) the half-word item counts from the B and Y designators (16 bits), left-shifted 5 places, to the program address of this instruction.

If the specified compare condition is not met, the instructions will continue execution at the next sequential instruction.

If any of the following conditions occur, the operation of these instructions is undefined.

- G bit 0 = 1 and G bit 3 = 1.
- G bit 3 = 1 for B2, B3, B4, and B5.
- G bit 5 = 0 and G bit 6 = 1.

The CDC FMP has expanded capabilities for the B0 through B5 instructions implemented by means of G bit 0 through 3 combinations.

(continued)

3.2.1.182 (Cont.)

```
B0  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) EQ (Z)
B1  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) NE (Z)
B2  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) GE (Z)
B3  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) LT (Z)
B4  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) LE (Z)
B5  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) GT (Z)
```

If bit 0 of the G designator is cleared/set, registers A, X, Y, C, and Z are 64/32 bits respectively. Register B is not used and must be set to zero.

G bit 1 = 0 and G bit 2 = 1

These instructions are executed in 5 steps of which the first four (compare) steps are identical to the first four steps described for B0 through B5 instructions with G bits 1 and 2 equal to zero (compare branch).

If the specified compare condition is met, the instruction performs as follows:

> Store into register Y and 64-bit quantity (32-bit if G bit 0 = 1) 000---001 and continue execution at the next sequential instruction.

If the specified compare condition is not met, the instruction performs as follows:

> Store into register Y and 64-bit quantity (32-bit if G bit 0 = 1) 000---000 and continue execution at the next sequential instruction.

If any of the following conditions occur, the operation of these instructions is undefined:

- G bit 0 = 1 and G bit 3 = 1.
- G bit 3 = 1 for B2, B3, B4, and B5.
- G bit 5 = 1, G bit 6 = 1 or G bit 7 = 1.
- The C designator is equal to the Z designator.

(continued)

3.2.1.182 (Cont.)

```
B0  C  E  BR  COMPARE F.P., BRANCH IF (A) EQ (X)
B1  C  E  BR  COMPARE F.P., BRANCH IF (A) NE (X)
B2  C  E  BR  COMPARE F.P., BRANCH IF (A) GE (X)
B3  C  E  BR  COMPARE F.P., BRANCH IF (A) LT (X)
B4  C  E  BR  COMPARE F.P., BRANCH IF (A) LE (X)
B5  C  E  BR  COMPARE F.P., BRANCH IF (A) GT (X)
```

If bit 0 of the G designator is cleared/set, registers A and X are 64/32 bits, respectively. Registers B and Y are always 64 bits. Registers C and Z are not used and must be set to zero.

G bit 1 = 1 and G bit 2 = 0

These instructions compare the two floating-point operands from registers A and X according to the floating-point compare rules in Section 3.1.4.5.

If the specified compare condition is met, the instructions perform as follows:

- G bit 5 = 0 — Branch to the address formed by adding the half-word item count from register Y, left-shifted 5 places, to the base address from register B.
- G bit 5 = 1 — Branch to the address formed by adding (G bit 6 = 0) or subtracting (G bit 6 = 1) the half-word item counts from the B and Y designators 16 bits), left-shifted 5 places, to the program address of this instruction.

If the specified compare condition is not met, the instructions will continue execution at the next sequential instruction.

If any of the following conditions occur, the operation of these instructions is undefined:

- G bit 3 = 1, G bit 4 = 1 or G bit 7 = 1.
- Designator Z and/or C not equal to zero.
- G bit 5 = 0 and G bit 6 = 1.

Data flags: bits 46 and 58

(continued)

3.2.1.182 (Cont.)

```
B0  C  E  NT  COMPARE F.P, SET CONDITION IF (A) EQ (X)
B1  C  E  NT  COMPARE F.P, SET CONDITION IF (A) NE (X)
B2  C  E  NT  COMPARE F.P, SET CONDITION IF (A) GE (X)
B3  C  E  NT  COMPARE F.P, SET CONDITION IF (A) LT (X)
B4  C  E  NT  COMPARE F.P, SET CONDITION IF (A) LE (X)
B5  C  E  NT  COMPARE F.P, SET CONDITION IF (A) GT (X)
```

If bit 0 of the G designator is cleared/set, registers A, X, and Y are 64/32 bits, respectively. Registers B, C, and Z are not used and must be set to zero.

G bit 1 = 1 and G bit 2 = 1

These instructions compare the two floating-point operands from registers A and X according to the floating-point compare rules in Section 3.1.4.5.

If the specified compare condition is met the instruction performs as follows:

> Store into register Y and 64-bit quantity (32-bit if G bit 0 = 1) 000---000 and continue execution at the next sequential instruction.

If the specified compare condition is not met, the instruction performs as follows:

> Store into register Y the 64-bit quantity (32-bit if G bit 0 = 1) 000---001 and continue execution at the next sequential instruction.

If any of the following conditions occur, the operation of these instructions is undefined:

- Any one of G bits 3 through 7 is set.
- Designators B, Z and/or C are not equal to zero.

Data flags: bits 46 and 58

3.2.1.183 B6 5 NA BR BRANCH TO IMMEDIATE ADDRESS;
(R) + I(48) BITS)

The right-most 48 bits of register R contain an item count of half-words. The right-most 48 bits of the instruction word contain an immediate operand which is used as a base address. An unconditional branch is taken to the branch address formed by adding the item count to the base address (the item count is shifted left 5 places before the addition and overflow, if any, is ignored).

A direct branch is taken to the base address from the instruction word if the R designator is zero or if the right-most 43 bits of register R are zeros.

3.2.1.184 B7 ILLEGAL

3.2.1.185 B8 ILLEGAL

3.2.1.186 B9 ILLEGAL

3.2.1.187 BA ILLEGAL

3.2.1.188 BB ILLEGAL

3.2.1.189 BC ILLEGAL

3.2.1.190 BD ILLEGAL

3.2.1.191 BE 5 64 IN ENTER (R) WITH I(48 BITS)

Clear register R and transfer the right-most 48 bits of this instruction to the right-most 48 bits of register R.

3.2.1.192 BF 5 64 IN INCREASE (R) BY I(48 BITS)

Replace the right-most 48 bits of register R by the sum of those bits and the right-most 48 bits of this instruction word. Arithmetic overflow is ignored.

3.2.1.193 C0 ILLEGAL

3.2.1.194 C1 ILLEGAL

3.2.1.195 C2 ILLEGAL

3.2.1.196 C3 ILLEGAL

3.2.1.197 C4 ILLEGAL

3.2.1.198 C5 ILLEGAL

3.2.1.199 C6 ILLEGAL

3.2.1.200 C7 ILLEGAL

3.2.1.201 C8 ILLEGAL

3.2.1.202 C9 ILLEGAL

3.2.1.203 CA ILLEGAL

3.2.1.204 CB ILLEGAL

3.2.1.205 CC ILLEGAL

3.2.1.206 CD 5 32 IN HALF-WORD ENTER (R) WITH I(24 BITS)

Clear register R and transfer the right-most 24 bits of this instruction to the right-most 24 bits of register R.

3.2.1.207 CE 5 32 IN HALF-WORD INCREASE (R) BY I(24 BITS)

Replace the right-most 24 bits of register R by the sum of those bits and the right-most 24 bits of this instruction word. Arithmetic overflow is ignored.

3.2.1.208 CF ILLEGAL

3.2.1.209 D0 ILLEGAL

3.2.1.210 D1 ILLEGAL

3.2.1.211 D2 ILLEGAL

3.2.1.212 D3 ILLEGAL

3.2.1.213 D4 ILLEGAL

3.2.1.214 D5 ILLEGAL

3.2.1.215 D6 ILLEGAL

3.2.1.216 D7 ILLEGAL

3.2.1.217 D8 ILLEGAL

3.2.1.218 D9 ILLEGAL

3.2.1.219 DA ILLEGAL

3.2.1.220 DB ILLEGAL

3.2.1.221 DC ILLEGAL

3.2.1.222 DD ILLEGAL

3.2.1.223 DE ILLEGAL

3.2.1.224 DF ILLEGAL

3.2.1.225 E0 ILLEGAL

3.2.1.226 E1 ILLEGAL

3.2.1.227 E2 ILLEGAL

3.2.1.228 E3 ILLEGAL

3.2.1.229 E4 ILLEGAL

3.2.1.230 E5 ILLEGAL

3.2.1.231 E6 ILLEGAL

3.2.1.232 E7 ILLEGAL

3.2.1.233 E8 ILLEGAL

3.2.1.234 E9 ILLEGAL

3.2.1.235	EA	ILLEGAL
3.2.1.236	EB	ILLEGAL
3.2.1.237	EC	ILLEGAL
3.2.1.238	ED	ILLEGAL
3.2.1.239	EE	ILLEGAL
3.2.1.240	EF	ILLEGAL
3.2.1.241	F0	ILLEGAL
3.2.1.242	F1	ILLEGAL
3.2.1.243	F2	ILLEGAL
3.2.1.244	F3	ILLEGAL
3.2.1.245	F4	ILLEGAL
3.2.1.246	F5	ILLEGAL
3.2.1.247	F6	ILLEGAL
3.2.1.248	F7	ILLEGAL
3.2.1.249	F8	ILLEGAL
3.2.1.250	F9	ILLEGAL
3.2.1.251	FA	ILLEGAL
3.2.1.252	FB	ILLEGAL

3.2.1.253 FC ILLEGAL

3.2.1.254 FD ILLEGAL

3.2.1.255 FE ILLEGAL

3.2.1.256 FF ILLEGAL

3.2.2 Instruction Execution Times

Instruction execution times are to be included in the appropriate machine description specifications. See Section 2.0.

4.0 TEST REQUIREMENTS (not applicable)

5.0 PREPARATION FOR DELIVERY (not applicable)

6.0 NOTES

6.1 ASCII/EBCDIC Reference Charts

The following table defines the control characters used in the ASCII Reference Chart.

NUL	Null	DLE	Data Link Escape (CC)
SOH	Start of Heading (CC)	DC1	Device Control 1
STX	Start of Text (CC)	DC2	Device Control 2
ETX	End of Text (CC)	DC3	Device Control 3
EOT	End of Transmission (CC)	DC4	Device Control 4 (Stop)
ENQ	Enquiry (CC)	NAK	Negative Acknowledge (CC)
ACK	Acknowledge (CC)	SYN	Synchronous Idle (CC)
BEL	Bell (audible or attention signal)	ETB	End of Transmission Block (CC)
BS	Backspace (FE)	CAN	Cancel
HT	Horizontal Tabulation (punched card skip (FE)	EM	End of Medium
LF	Line Feed (FE)	SUB	Substitute
VT	Vertical Tabulation (FE)	ESC	Escape
FF	Form Feed (FE)	FS	File Separator (IS)
CR	Carriage Return (FE)	GS	Group Separator (IS)
SO	Shift Out	RS	Record Separator (IS)
SI	Shift In	US	Unit Separator (IS)
		DEL	Delete[1]

NOTE: (CC) Communication Control
(FE) Format Effector
(IS) Information Separator

[1] In the strict sense, DEL is not a control character.

ASCII REFERENCE CHART

b_8 b_7 b_6 b_5 →		0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
b_4 b_3 b_2 b_1 ↓	COL. → / ROW ↓	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p								
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q								
0 0 1 0	2	STX	DC2	"	2	B	R	b	r								
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s								
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t								
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u								
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v								
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w								
1 0 0 0	8	BS	CAN	(	8	H	X	h	x								
1 0 0 1	9	HT	EM	)	9	I	Y	i	y								
1 0 1 0	10 (A)	LF	SUB	*	:	J	Z	j	z								
1 0 1 1	11 (B)	VT	ESC	+	;	K	[	k	{								
1 1 0 0	12 (C)	FF	FS	,	<	L	\	l	¦								
1 1 0 1	13 (D)	CR	GS	−	=	M	]	m	}								
1 1 1 0	14 (E)	SO	RS	.	>	N	^	n	~								
1 1 1 1	15 (F)	SI	US	/	?	O	_	o	DEL								EO

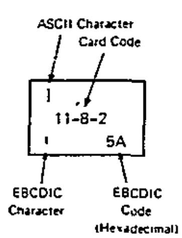
EBCDIC REFERENCE CHART

$b_8\ b_7\ b_6\ b_5$		0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
$b_4\ b_3\ b_2\ b_1$	COL. → / ROW ↓	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0 0 0 0	0	NUL	DLE			SP	&	—						{	}	\	0
0 0 0 1	1	SOH	DC1					/		a	j	~		A	J		1
0 0 1 0	2	STX	DC2		SYN					b	k	s		B	K	S	2
0 0 1 1	3	ETX	DC3							c	1	t		C	L	T	3
0 1 0 0	4									d	m	u		D	M	U	4
0 1 0 1	5	HT		LF						e	n	v		E	N	V	5
0 1 1 0	6		BS	ETB						f	o	w		F	O	W	6
0 1 1 1	7	DEL		ESC	EOT					g	p	x		G	P	X	7
1 0 0 0	8		CAN							h	q	y		H	Q	Y	8
1 0 0 1	9		EM						\	i	r	z		I	R	Z	9
1 0 1 0	10 (A)					[	]	¦	:								
1 0 1 1	11 (B)	VT				.	$	,	#								
1 1 0 0	12 (C)	FF	FS		DC4	<	*	%	@								
1 1 0 1	13 (D)	CR	GS	ENQ	NAK	(	)	_	'								
1 1 1 0	14 (E)	SO	RS	ACK		+	;	>	=								
1 1 1 1	15 (F)	SI	US	BEL	SUB	!	^	?	"								EO

AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) WITH PUNCHED CARD CODES AND EBCDIC TRANSLATION

	b8 b7 b6 b5	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
b4 b3 b2 b1	COL / ROW	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0 0 0 0	0	NUL 12-0-9-8-1 NUL 00	DLE 12-11-9-8-1 DLE 10	SP no punch SP 40	0 0 0 F0	@ 8-4 @ 7C	P 11-7 P D7	\` 8-1 79	p 12-11-7 p 97	11-0-9-8-1 DS 20	12-11-0-9-8-1 30	12-0-9-1 41	12-11-9-8 58	12-11-0-9-6 76	12-11-8-7 9F	12-11-0-8 B8	12-11-9-8-4 DC
0 0 0 1	1	SOH 12-9-1 SOH 01	DC1 11-9-1 DC1 11	! 12-8-7 4F	1 1 1 F1	A 12-1 A C1	Q 11-8 Q D8	a 12-0-1 a 81	q 12-11-8 q 98	0-9-1 SOS 21	9-1 31	12-0-9-2 42	11-8-1 59	12-11-0-9-7 77	11-0-8-1 A0	12-11-0-9 B9	12-11-9-8-5 DD
0 0 1 0	2	STX 12-9-2 STX 02	DC2 11-9-2 DC2 12	" 8-7 7F	2 2 2 F2	B 12-2 B C2	R 11-9 R D9	b 12-0-2 b 82	r 12-11-9 r 99	0-9-2 FS 22	11-9-8-2 CC 1A	12-0-9-3 43	11-0-9-2 62	12-11-0-9-8 78	11-0-8-2 AA	12-11-0-8-2 BA	12-11-9-8-6 DE
0 0 1 1	3	ETX 12-9-3 ETX 03	DC3 11-9-3 TM 13	# 8-3 # 7B	3 3 3 F3	C 12-3 C C3	S 0-2 S E2	c 12-0-3 c 83	s 11-0-2 s A2	0-9-3 23	9-3 33	12-0-9-4 44	11-0-9-3 63	12-0-8-1 80	11-0-8-3 AB	12-11-0-8-3 BB	12-11-9-8-7 DF
0 1 0 0	4	EOT 9-7 EOT 37	DC4 9-8-4 DC4 3C	$ 11-8-3 $ 5B	4 4 4 F4	D 12-4 D C4	T 0-3 T E3	d 12-0-4 d 84	t 11-0-3 t A3	0-9-4 BYP 24	9-4 PN 34	12-0-9-5 45	11-0-9-4 64	12-0-8-2 8A	11-0-8-4 AC	12-11-0-8-4 BC	11-0-9-8-2 EA
0 1 0 1	5	ENQ 0-9-8-5 ENQ 2D	NAK 9-8-5 NAK 3D	% 0-8-4 % 6C	5 5 5 F5	E 12-5 E C5	U 0-4 U E4	e 12-0-5 e 85	u 11-0-4 u A4	11-9-5 NL 15	9-5 RS 35	12-0-9-6 46	11-0-9-5 65	12-0-8-3 8B	11-0-8-5 AD	12-11-0-8-5 BD	11-0-9-8-3 EB
0 1 1 0	6	ACK 0-9-8-6 ACK 2E	SYN 9-2 SYN 32	& 12 & 50	6 6 6 F6	F 12-6 F C6	V 0-5 V E5	f 12-0-6 f 86	v 11-0-5 v A5	12-9-6 LC 06	9-6 UC 36	12-0-9-7 47	11-0-9-6 66	12-0-8-4 8C	11-0-8-6 AE	12-11-0-8-6 BE	11-0-9-8-4 EC
0 1 1 1	7	BEL 0-9-8-7 BEL 2F	ETB 0-9-6 ETB 26	' 8-5 ' 7D	7 7 7 F7	G 12-7 G C7	W 0-6 W E6	g 12-0-7 g 87	w 11-0-6 w A6	11-9-7 IL 17	12-9-8 GE 08	12-0-9-8 48	11-0-9-7 67	12-0-8-5 8D	11-0-8-7 AF	12-11-0-8-7 BF	11-0-9-8-5 ED
1 0 0 0	8	BS 11-9-6 BS 16	CAN 11-9-8 CAN 18	(12-8-5 (4D	8 8 8 F8	H 12-8 H C8	X 0-7 X E7	h 12-0-8 h 88	x 11-0-7 x A7	0-9-8 28	9-8 38	12-8-1 49	11-0-9-8 68	12-0-8-6 8E	12-11-0-8-1 B0	12-0-9-8-2 CA	11-0-9-8-6 EE
1 0 0 1	9	HT 12-9-5 HT 05	EM 11-9-8-1 EM 19	) 11-8-5) 5D	9 9 9 F9	I 12-9 I C9	Y 0-8 Y E8	i 12-0-9 i 89	y 11-0-8 y A8	0-9-8-1 29	9-8-1 39	12-11-9-1 51	0-8-1 69	12-0-8-7 8F	12-11-0-1 B1	12-0-9-8-3 CB	11-0-9-8-7 EF
1 0 1 0	10 (A)	LF 0-9-5 LF 25	SUB 9-8-7 SUB 3F	* 11-8-4 5C	: 8-2 7A	J 11-1 J D1	Z 0-9 Z E9	j 12-11-1 j 91	z 11-0-9 A9	0-9-8-2 SM 2A	9-8-2 3A	12-11-9-2 52	12-11-0 70	12-11-8-1 90	12-11-0-2 B2	12-0-9-8-4 CC	12-11-0-9-8-7 (LVM) FA
1 0 1 1	11 (B)	VT 12-9-8-3 VT 0B	ESC 0-9-7 ESC 27	+ 12-8-6 + 4E	; 11-8-6 5E	K 11-2 K D2	[12-8-2 ¢ 4A	k 12-11-2 k 92	{ 12-0 C0	0-9-8-3 CU2 2B	9-8-3 CU3 3B	12-11-9-3 53	12-11-0-9-1 71	12-11-8-2 9A	12-11-0-3 B3	12-0-9-8-5 CD	12-11-0-9-8-3 FB
1 1 0 0	12 (C)	FF 12-9-8-4 FF 0C	FS 11-9-8-4 IFS 1C	, 0-8-3 6B	< 12-8-4 < 4C	L 11-3 L D3	\\ 0-8-2 E0	l 12-11-3 93	| 12-11 | 6A	0-9-8-4 2C	12-9-4 PF 04	12-11-9-4 54	12-11-0-9-2 72	12-11-8-3 9B	12-11-0-4 B4	12-0-9-8-6 CE	12-11-0-9-8-4 FC
1 1 0 1	13 (D)	CR 12-9-8-5 CR 0D	GS 11-9-8-5 IGS 1D	- 11 60	= 8-6 7E	M 11-4 M D4	] 11-8-2 ! 5A	m 12-11-4 m 94	} 11-0 D0	12-9-8-1 RLF 09	11-9-4 RES 14	12-11-9-5 55	12-11-0-9-3 73	12-11-8-4 9C	12-11-0-5 B5	12-0-9-8-7 CF	12-11-0-9-8-5 FD
1 1 1 0	14 (E)	SO 12-9-8-6 SO 0E	RS 11-9-8-6 IRS 1E	. 12-8-3 4B	> 0-8-6 > 6E	N 11-5 N D5	^ 11-8-7 ¬ 5F	n 12-11-5 n 95	~ 11-0-1 A1	12-9-8-2 SMM 0A	9-8-6 3E	12-11-9-6 56	12-11-0-9-4 74	12-11-8-5 9D	12-11-0-6 B6	12-11-9-8-2 DA	12-11-0-9-8-6 FE
1 1 1 1	15 (F)	SI 12-9-8-7 SI 0F	US 11-9-8-7 IUS 1F	/ 0-1 61	? 0-8-7 ? 6F	O 11-6 O D6	_ 0-8-5 6D	o 12-11-6 o 96	DEL 12-9-7 DEL 07	11-9-8-3 CU1 1B	11-0-9-1 E1	12-11-9-7 57	12-11-0-9-5 75	12-11-8-6 9E	12-11-0-7 B7	12-11-9-8-3 DB	EO 12-11-0-9-8-7 FF

LEGEND

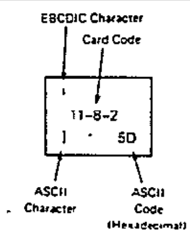


ORIGINAL PAGE IS OF POOR QUALITY

EXTENDED BINARY CODED DECIMAL INTERCHANGE CODE (EBCDIC) WITH PUNCHED CARD CODES AND ASCII TRANSLATION

BITS 4 5 6 7	BITS 0 1 2 3 / 1ST HEX / 2ND	0 0 0 0 0	0 0 0 1 1	0 0 1 0 2	0 0 1 1 3	0 1 0 0 4	0 1 0 1 5	0 1 1 0 6	0 1 1 1 7	1 0 0 0 8	1 0 0 1 9	1 0 1 0 A (10)	1 0 1 1 B (11)	1 1 0 0 C (12)	1 1 0 1 D (13)	1 1 1 0 E (14)	1 1 1 1 F (15)
0 0 0 0	0	NUL 12-0-9-8-1 NUL 00	DLE 12-11-9-8-1 DLE 10	DS 11-0-9-8-1 80	12-11-0-9-8-1 90	SP no punch SP 20	& 12 & 26	- 11 - 2D	12-11-0 8A	12-0-8-1 C3	12-11-8-1 CA	11-0-8-1 D1	12-11-0-8-1 D8	{ 12-0 { 7B	} 11-0 } 7D	0-8-2 5C	0 0 0 30
0 0 0 1	1	SOH 12-9-1 SOH 01	DC1 11-9-1 DC1 11	SOS 0-9-1 81	9-1 91	12-0-9-1 A0	12-11-9-1 A9	0-1 / 2F	12-11-0-9-1 8B	a 12-0-1 a 61	j 12-11-1 j 6A	~ 11-0-1 ~ 7E	12-11-0-1 D9	A 12-1 A 41	J 11-1 J 4A	11-0-9-1 9F	1 1 1 31
0 0 1 0	2	STX 12-9-2 STX 02	DC2 11-9-2 DC2 12	FS 0-9-2 82	SYN 9-2 SYN 16	12-0-9-2 A1	12-11-9-2 AA	11-0-9-2 B2	12-11-0-9-2 8C	b 12-0-2 b 62	k 12-11-2 k 6B	s 11-0-2 s 73	12-11-0-2 DA	B 12-2 B 42	K 11-2 K 4B	S 0-2 S 53	2 2 2 32
0 0 1 1	3	ETX 12-9-3 ETX 03	TM 11-9-3 DC3 13	0-9-3 83	9-3 93	12-0-9-3 A2	12-11-9-3 AB	11-0-9-3 B3	12-11-0-9-3 8D	c 12-0-3 c 63	l 12-11-3 l 6C	t 11-0-3 t 74	12-11-0-3 DB	C 12-3 C 43	L 11-3 L 4C	T 0-3 T 54	3 3 3 33
0 1 0 0	4	PF 12-9-4 9C	RES 11-9-4 9D	BYP 0-9-4 84	PN 9-4 94	12-0-9-4 A3	12-11-9-4 AC	11-0-9-4 B4	12-11-0-9-4 8E	d 12-0-4 d 64	m 12-11-4 m 6D	u 11-0-4 u 75	12-11-0-4 DC	D 12-4 D 44	M 11-4 M 4D	U 0-4 U 55	4 4 4 34
0 1 0 1	5	HT 12-9-5 HT 09	NL 11-9-5 85	LF 0-9-5 LF 0A	RS 9-5 95	12-0-9-5 A4	12-11-9-5 AD	11-0-9-5 B5	12-11-0-9-5 8F	e 12-0-5 e 65	n 12-11-5 n 6E	v 11-0-5 v 76	12-11-0-5 DD	E 12-5 E 45	N 11-5 N 4E	V 0-5 V 56	5 5 5 35
0 1 1 0	6	LC 12-9-6 86	BS 11-9-6 BS 08	ETB 0-9-6 ETB 17	UC 9-6 96	12-0-9-6 A5	12-11-9-6 AE	11-0-9-6 B6	12-11-0-9-6 C0	f 12-0-6 f 66	o 12-11-6 o 6F	w 11-0-6 w 77	12-11-0-6 DE	F 12-6 F 46	O 11-6 O 4F	W 0-6 W 57	6 6 6 36
0 1 1 1	7	DEL 12-9-7 DEL 7F	IL 11-9-7 87	ESC 0-9-7 ESC 1B	EOT 9-7 EOT 04	12-0-9-7 A6	12-11-9-7 AF	11-0-9-7 B7	12-11-0-9-7 C1	g 12-0-7 g 67	p 12-11-7 p 70	x 11-0-7 x 78	12-11-0-7 DF	G 12-7 G 47	P 11-7 P 50	X 0-7 X 58	7 7 7 37
1 0 0 0	8	GE 12-9-8 97	CAN 11-9-8 CAN 18	0-9-8 88	9-8 98	12-0-9-8 A7	12-11-9-8 B0	11-0-9-8 B8	12-11-0-9-8 C2	h 12-0-8 h 68	q 12-11-8 q 71	y 11-0-8 y 79	12-11-0-8 E0	H 12-8 H 48	Q 11-8 Q 51	Y 0-8 Y 59	8 8 8 38
1 0 0 1	9	RLF 12-9-8-1 8D	EM 11-9-8-1 EM 19	0-9-8-1 89	9-8-1 99	12-8-1 A8	11-8-1 B1	0-8-1 B9	` 8-1 60	12-0-9 i 69	r 12-11-9 r 72	z 11-0-9 z 7A	12-11-0-9 E1	I 12-9 I 49	R 11-9 R 52	Z 0-9 Z 5A	9 9 9 39
1 0 1 0	A (10)	SMM 12-9-8-2 8E	CC 11-9-8-2 92	SM 0-9-8-2 8A	9-8-2 9A	¢ 12-8-2 [5B	! 11-8-2] 5D	¦ 12-11 ¦ 7C	8-2 3A	12-0-8-2 C4	12-11-8-2 CB	11-0-8-2 D2	12-11-0-8-2 E2	12-0-9-8-2 E8	12-11-9-8-2 EE	11-0-9-8-2 F4	(LVM) 12-11-0-9-8-2 FA
1 0 1 1	B (11)	VT 12-9-8-3 VT 0B	CU1 11-9-8-3 8F	CU2 0-9-8-3 8B	CU3 9-8-3 9B	12-8-3 2E	$ 11-8-3 $ 24	0-8-3 , 2C	# 8-3 # 23	12-0-8-3 C5	12-11-8-3 CC	11-0-8-3 D3	12-11-0-8-3 E3	12-0-9-8-3 E9	12-11-9-8-3 EF	11-0-9-8-3 F5	12-11-0-9-8-3 FB
1 1 0 0	C (12)	FF 12-9-8-4 FF 0C	IFS 11-9-8-4 FS 1C	0-9-8-4 8C	DC4 9-8-4 DC4 14	< 12-8-4 < 3C	* 11-8-4 2A	% 0-8-4 % 25	@ 8-4 @ 40	12-0-8-4 C6	12-11-8-4 CD	11-0-8-4 D4	12-11-0-8-4 E4	⑀ 12-0-9-8-4 EA	12-11-9-8-4 F0	⑁ 11-0-9-8-4 F6	12-11-0-9-8-4 FC
1 1 0 1	D (13)	CR 12-9-8-5 CR 0D	IGS 11-9-8-5 GS 1D	ENQ 0-9-8-5 ENQ 05	NAK 9-8-5 NAK 15	(12-8-5 (28	) 11-8-5) 29	0-8-5 5F	' 8-5 27	12-0-8-5 C7	12-11-8-5 CE	11-0-8-5 D5	12-11-0-8-5 E5	12-0-9-8-5 EB	12-11-9-8-5 F1	11-0-9-8-5 F7	12-11-0-9-8-5 FD
1 1 1 0	E (14)	SO 12-9-8-6 SO 0E	IRS 11-9-8-6 RS 1E	ACK 0-9-8-6 ACK 06	9-8-6 9E	12-8-6 + 2B	11-8-6 3B	⌐ 0-8-6 > 3E	8-6 3D	12-0-8-6 C8	12-11-8-6 CF	11-0-8-6 D6	12-11-0-8-6 E6	⑂ 12-0-9-8-6 EC	12-11-9-8-6 F2	11-0-9-8-6 F8	12-11-0-9-8-6 FE
1 1 1 1	F (15)	SI 12-9-8-7 SI 0F	IUS 11-9-8-7 US 1F	BEL 0-9-8-7 BEL 07	SUB 9-8-7 SUB 1A	! 12-8-7 ! 21	¬ 11-8-7 ^ 5E	? 0-8-7 ? 3F	8-7 22	12-0-8-7 C9	12-11-8-7 D0	11-0-8-7 D7	12-11-0-8-7 E7	12-0-9-8-7 ED	12-11-9-8-7 F3	11-0-9-8-7 F9	12-11-0-9-8-7 EO FF

LEGEND



APPENDIX A

A1.0 SCOPE

The intent of this Appendix is to provide additional information regarding some of the characteristics of the CDC FMP. Further information can be found in other appropriate specifications. (See Section 2.0, Applicable Documents).

A2.0 SELF-MODIFYING PROGRAMS

The use of self-modifying programs is not allowed. The following rules which would have to be followed illustrate why this must be true.

The following rules apply to all programs:

1. The twenty-four 64-bit words before (having addresses lower than) and the thirty-two 64-bit words after (having addresses higher than) the current instruction word shall not be modified by the current instruction.

2. The twenty-four instructions before (in terms of order of execution) and the thirty-two instructions after (in terms of order of execution) the current instruction word shall not be modified by the current instruction.

3. The store into Main Memory for the 13, 5F, and 7F instructions may not take place before the execution of the next instruction in sequence. Therefore, if these instructions are used to modify code, it is difficult to guarantee that the store has taken place before the execution of that code. There are three procedures to guarantee that the store has taken place prior to execution of the intended modified code.

   a. The execution of any instruction which references Main Memory with the exception of the 12, 13, 32, 5E, 5F, 7E, and 7F instructions. These instructions must be executed between the store instruction which modifies the code and the use of that modified code.

(continued)

A2.0 (Cont.)

b. The execution of the conditional branch feature of the 32 instruction between the store instruction which modifies the code and the use of that modified code.

c. Execution of a load instruction (12, 5E, or 7E) followed by a transmit (78) instruction where the source register for the 78 instruction conflicts with the destination register for the load instruction. These instructions must be executed between the store instruction which modifies the code and the use of that modified code.

The instructions referenced in a., b., and c. above must be executed from addresses at least four swords before or at least three swords after the modified code.

A3.0 INSTRUCTION STACK

Each machine has a different size instruction stack thus program optimization must be approached with different parameters. Further information is contained in the appropriate execution timing specification.

<u>Number of Words in Instruction Stack</u>

CDC STAR-1B	1	64-bit word
CDC STAR-100	32	64-bit words
CDC STAR-100A	128	32-bit words
CDC FMP	128	32-bit words

A4.0 (N/A)

A5.0 VECTOR FORMATS

In the CDC FMP, a vector is defined as a contiguous set of bits, bytes, or floating-point operands. The contiguous set of bits or bytes is called a string, while the contiguous set of floating-point elements is called an array.

(continued)

A5.0 (Cont.)

Operands are used in the following vector formats:

Array - a counted, variable-length, contiguous, floating-point operand field. Vector operations can be performed on defined fields consisting entirely of 32-bit operands or entirely of 64-bit operands.

Index List - a counted data array of integer values in floating-point format.

A6.0 DATA FLAG BRANCH

The automatic data flag branch can occur up to 35 instructions after the instruction which caused it. The point at which the branch occurs can vary between-- executions of the same program as a result of the asynchronous I/O activity affecting the load/store operations.

The following points pertain to the use of the data flag register:

1. The contents of the DFR as stored into the register file by a 3B instruction will reflect all previous activity on it. Also, activity prior to the 3B instruction will not affect the new contents of the DFR.

2. ADFB's caused by a 3B instruction or any instruction previous to it may occur after the next one or two instructions, but no later.

3. Sampling or altering a data flag bit with a 33 instruction may occur out of sequence with a previous pipeline instruction up to 35 instructions earlier.

4. If a 33 instruction alters a bit which causes an ADFB, the branch may occur up to two instructions later, regardless

(continued)

A6.0 (Cont.)

of the fact that all pipeline instructions previous to it may have finished. Again, if the ADFB is also contingent on the completion of a pipeline instruction, the automatic data flag branch may occur up to 35 instructions after the instruction which caused it.

When registers 1, 2, or 4 in the FMP register file are altered by an instruction, and this instruction is followed by an automatic flag branch or illegal monitor instruction branch, the store operation may happen out of sequence with the branch operation. Thus, for example, if a 7E instruction loads register 4 with a certain value, and this instrucion is followed by an illegal monitor mode instruction, the automatic branch will be to the address specified by either the old or new contents of register 4, depending on the timing of the 7E and the instruction stream.

DIVISION 3

STANDARD PRODUCT

SYSTEM COMPONENTS

CYBER 170/MODEL 175-100 COMPUTER SYSTEM

The CDC CYBER 170/Model 175-100 computer system is a multipurpose computing system that provides real-time/time-critical network, commercial, data management, and scientific capabilities. It can serve as an entry level processor to the Model 175 family of computers, and is field upgradable to a Model 175-200 or Model 175-300 system.

The Model 175-100 computing system has one large central processor, accessible through central memory, and a group of 10 to 20 (optional) peripheral processors. Each of these peripheral processors has a separate memory and can execute programs independently of each other or of the central processor. Through the exchange jump feature and central memory communication, these peripheral processors control the central processor. They communicate with themselves through central memory and the input/output channels.

In solving a problem, one or more peripheral processors are used for high-speed information transfer in and out of the system, and to provide operator control. A number of problems can be in operation concurrently by time-sharing the central processor. Further concurrency is obtained with the central processor by parallel action of various functional units.

Central memory is organized into logically independent banks. Many banks can be in operation simultaneously, thereby minimizing execution time. The multiple operation modes of all segments of the computer, in combination with high-speed circuits, produce a very high overall computing speed.

HIGHLIGHTS

Central Processor Characteristics

- Nine arithmetic functional units for concurrent operations
- 24 operating registers
  - 8 operand (60-bit)
  - 8 address (18-bit)
  - 8 increment (18-bit)
- Instruction stack which holds up to twelve 60-bit instruction words with 2-word look-ahead capability
- Computation in floating-point and fixed point, single and double precision
- Central exchange jump

Peripheral Processing Subsystem Characteristics

- Emitter-Coupled Logic (ECL) integrated circuits
- 10 to 20 peripheral processor configurations available (characteristics as listed are per processor)
- Semiconductor memory of 4096 12-bit words plus one parity bit per word (odd parity)
- 12 or 24 input/output channels
  - all channels common to all processors
  - transfer rate per channel 2 MHz (optionally 1 MHz) in 12-bit words.
  - all channels can be active simultaneously
  - all channels 12-bit plus parity (odd) bidirectional
- Major cycle time of 500 nanoseconds
- Computation in fixed point
- Time-shared access to central memory
- 64-instruction repertoire
- Status and control register monitors error conditions (maintenance aid)

Central Memory Characteristics

- Semiconductor memory model options listed with associated capacity of 60-bit words plus eight error correction bits per word.

Model 175-108	131,072 words
Model 175-112	196,608 words
Model 175-116	262,144 words

- Memory organized in logically independent banks of words with corresponding multiphasing of banks (maximum memory size of 16 banks)

- Transfer rate of up to one word each 50 nanoseconds in phased operation

CENTRAL PROCESSOR DESCRIPTION

The central processor, composed of a central processing unit, nine separate functional units which operate in parallel, and the central memory control, communicates only with central memory. It is isolated from the peripheral processors and is thus free to carry on computation unencumbered by input/output requirements. The nine arithmetic and logical units (floating add, normalize, long add, increment, shift, multiply, divide, boolean, and population count) execute the arithmetic, manipulative, and logical operations. The central memory control directs the arithmetic operations and provides the interface between the functional units and central memory. It also performs instruction retrieving, address preparation, memory protection, and data retrieving and storing. The central memory control provides for orderly flow of data between central memory and the requesting elements of the system.

The following are the general categories of central processor instructions:

- branch
- central exchange jump
- fixed-point arithmetic
- floating-point arithmetic
- extended core storage communication
- increment
- logical
- monitor, stop
- pass
- shift

PERIPHERAL PROCESSING SUBSYSTEM DESCRIPTION

The peripheral processing subsystem consists of 10 identical units that operate independently and simultaneously as stored-program computers. Many programs can be running at one time or a combination of processors can be involved in one problem which may require a variety of input/output tasks as well as use of the central memory and the central processor.

The peripheral processors act as system control computers and input/output processors. This permits the central processor to continue computations while the peripheral processors do the slower input/output and supervisory operations. Each processor has a 12-bit (plus one parity bit), random-access memory (independent of central memory) with a cycle time of 500 nanoseconds.

The following are the general categories of peripheral processor instructions:

- arithmetic
- central processor and central memory communications
- data transmission
- input/output
- logical
- no operation
- replace
- shift

Exchange Jump

The exhange jump interrupts an executing central processor program, saves a "snapshot" status of the program (registers, and so forth), and initiates execution of another program. The Model 175-100 provides four types of exchange jump instructions: one exchange jump initiated by the central processor, and three types of peripheral processor initiated exchange jumps.

One type of peripheral processor exchange jump is an unconditional jump. The other types are similar, except that each is dependent on a different condition of a hardware flag. However, in each case the contents of the operating and control registers are moved into storage, and the vacated registers are then loaded with the exchange package information from central memory. This permits a new program to be started by the central processor, and it maintains the information needed to resume the program that was executing.

Input/Output Channels

All processors communicate with external equipment and each other using the independent, bidirectional input/output channels. The number of channels depends on the number of peripheral processors in the system. All channels are 12-bit (plus parity) and each can be connected to one or more external devices. Only one external equipment can use a channel at one time, but all channels can be simultaneously active. Data is transferred into or out of the system in 12-bit words at a maximum rate of two words per microsecond. As many as eight different types of external equipment, for example, magnetic tape controllers and card reader/punch controllers, can be connected to an input/output channel.

Display Controller/Console

The display controller provides digital and analog input information to control the various presentations on the display console which consists of a cathode-ray tube display and a keyboard. The console keyboard performs the functions of a typewriter and permits manual entry of data. Some of the operator actions at the console include:

- display of all registers
- dynamic display of any word or block of words in central memory
- execute single steps of a program

Data Channel Converters

Two data channel converters, included with the system, allow Control Data 3000 series type peripheral equipment to be attached to the Model 175-100 input/output channels.

MODEL 175 MEMORY DESCRIPTION

The CDC CYBER 70/Model 175-100 computer offers a hierarchical memory concept:

- Central memory
- Extended core storage

Central memory provides a fast, random-access storage for executing programs and data, while the large extended core storage is used for input/ouput buffering, containing large data arrays, and to support job swapping and other operating system service functions. Although extended core storage is optional, it is fully supported and recommended for optimum system performance.

Central Memory

The Model 175-100 central memory is composed of banks of 60-bit words of MOS storage with eight error correction bits per 60-bit word. The complete cycle time for one bank is 400 nanoseconds. The banks are phased so that successive addresses are in different banks to permit operation of central memory at much higher rates than the basic cycle time. The maximum transfer rate is one 60-bit word per 50 nanoseconds.

Central memory is available in sizes ranging from 131,072 words (8 banks) to 262,144 words (16 banks). The large number of banks in central memory minimizes memory access conflicts; therefore, central memory is highly effective for fetching instructions and randomly accessing data items at very high rates.

There are four access paths to central memory:

- Central processor/central memory
- Extended core storage/central memory
- One or two groups of peripheral processor/central memory

Central memory control (part of central processor) provides service to each of these access paths on a priority basis, queues access requests as necessary, and resolves any access conflicts. The control section also generates the eight error correction bits for each 60-bit word placed in central memory. When a word is read from central memory, manipulation of the error correction bits allows a 60-bit word containing a single-bit error to be corrected. Double-bit errors and most multiple-bit errors are detected but not corrected, and their occurrence is noted in the status and control register.

Extended Core Storage Subsystem

The optional extended core storage (ECS) subsystem is comprised of the extended core storage, its controller, and one or more distributive data paths which attach to input/output channels.

The extended core storage is composed of banks of 131,072 60-bit records of core storage. Eight 60-bit words are contained in a 488-bit physical extended core storage word. Each 60-bit word has an associated parity bit in the extended core storage word. The complete cycle time for one bank of ECS is 3.2 microseconds per 488-bit extended core storage word.

In multiple bank extended core storage subsystems, banks are phased such that consecutive 8-word records come from different banks. This phasing, combined with the wide (8-word) access span enables very fast transfers to or from ECS. After initial access, ECS can transfer at a rate of one 60-bit word per 100 nanoseconds. This gives a maximum rate of 600 million bits per second.

There are two access paths to extended core storage:

- Central memory to extended core storage
- Input/output channel(s) extended core storage using the distributive data paths

7030 CYBER EXTENDED CORE STORAGE

The CDC 7030 CYBER Extended Core Storage (ECS) is a random-access word-organized mass storage device. ECS, through its controller, communicates with one or more CDC CYBER 70 and CYBER 170 series computer systems using both central memory and input/output channels.

HIGHLIGHTS

- Storage capacity from 262,144 to 2,097,152 (60-bit) words, depending on the model
- Transfer rate up to 10,000,000 (60-bit) words per second
- Data paths between ECS and either central memory or as many as four input/output channels or both

DESCRIPTION

ECS with its controller serves as an extension to the computer central memory, providing expanded storage and serving as a high-speed input/output buffer. The ECS storage unit utilizes semiconductor technology and is available in 2-, 4-, 8-, or 16-bank configurations, each bank containing 131,070 words. The storage unit provides up to two million directly addressable 60-bit words. Eight 60-bit words are organized into a 488-bit data word in ECS (a parity bit is attached to each 60-bit word).

The ECS controller provides the data paths between central memory (60-bit) and the input/output channels (480-bit). The Distributive Data Path (DDP) consists of registers that connect ECS to input/output channels.

MODELS

The 7030 storage system is available in the following models:

Model	Feature
102	262K
104	524K
108	1048K
116	2097K

Distributive Data Path

The distributive data path provides a path for data flow between extended core storage and the peripheral processors. It allows fast peripheral processor access to data in extended core storage using an input/output channel, and greatly reduces the data traffic through the central memory. This reduces central memory conflicts and also reduces the overhead of the operating system. The distributive data path consists of 1-to-4 480-bit buffer registers. Each register connects to a standard input/output channel for maximum overlap of data transfer. All four registers share a single access to extended core storage. This arrangement allows up to four peripheral processors to transfer data simultaneously to extended core storage at the maximum rate of the channel. A 480-bit extended core storage word is assembled from a 40-word peripheral processor block in about 43 microseconds.

C-5

255X NETWORK PROCESSING UNIT

The CDC 255X Network Processing Unit (NPU) relieves the host computer (CDC 6000, CYBER 70, or CYBER 170) of data communication handling responsibilities, thereby freeing it for additional application functions. The NPU is comprised of modular hardware combined with specifically tailored software to provide efficient handling and processing of data communications in a network environment. This modularity affords a user the ability to expand the system to keep pace with increasing network communication requirements. Some of the features of the NPU include:

- Allows use of a wide range (in terms of characteristics and vendors) of terminal devices
- Resident (in NPU) Communication Control Program available in variations allowing intercommunication with NOS or NOS/BE
- Provides significant throughput capability
- Assumes data buffering requirements from the host computer

HIGHLIGHTS

Communications Processor

- 32K to 128K of 16-bit (plus 1 parity and 1 protect bit) MOS main memory words
- Main memory cycle time of 475 nanoseconds
- Microcode (program control) cycle time of 168 nanoseconds
- Eight memory addressing modes
- Instruction repertoire containing 14 instruction groups
- Memory word and region protection

- Main memory parity detection
- 15 levels of external interrupt and one internal interrupt
- Interrupt data channel transfer rate of 160,000 bytes per second

Channel Coupler

- Direct memory access transfer rate of 1,600,000 words per second

Multiplex Subsystem

- Input and output loop rates of 20,000,000 bits per second
- Cyclic check character generated on output loop and verified on input loop
- Demand driven communication line adapters
- Synchronous and asynchronous transmissions handled

COMMUNICATIONS PROCESSOR DESCRIPTION

The heart of the communications processor is the central processing unit. It is an integrated circuit, fully parallel unit which supplies the communications processor with basic stored program control and computational capabilities. It contains the facilities for fetching and storing data, provides microcode control for sequencing and executing instructions, has capabilities for processing arithmetic and logical data, and initiates and emulates the interrupts that constitute communication between input/output devices. The communications processor has a special set of communications-oriented instructions, enabling it to monitor connected communication lines and take appropriate alarm and safety action in the event of line or terminal faults.

The communications processor contains a maintenance console panel to implement the following:

- Display of register contents
- Display of memory locations
- Location of operational switches and indicators

CHANNEL COUPLER DESCRIPTION

The channel coupler controls the direct exchange of data between a peripheral processor unit in each of up to two host computers with optional coupler and the communication processor memory. Multiple couplers may share a host computer data channel or two couplers may be connected to a communication processor, thereby allowing multiple NPU or multiple host computer configurations for the purpose of redundancy.

The primary function of the channel coupler is to pass data blocks between the main computer and the communication processor with minimum software supervision. The coupler contains provisions for data protection and error detection.

MULTIPLEX SUBSYSTEM DESCRIPTION

The multiplexing subsystem uses a demand driven technique wherein data movement between a line and the communications processor's memory occurs at the instigation of a communication line adapter, not limited by scanning rate used in other techniques. The elements comprising the subsystem are described in the following paragraphs.

<u>Multiplex Loop</u>

The multiplex loop is the demand driven mechanism which gathers input data and status information from communications lines and distributes output data and control information to the lines. The loop is composed of two high-speed serial-transmission links, one for input, one for output. Data cycles through the links at a speed of 20 million bits per second. The loop is controlled by the communication processor via the multiplex loop interface adapter.

Multiplex Loop Interface Adapter (MLIA)

The MLIA interfaces between the loop multiplexer and either the communication processor or the multiplex loop controller. It converts bit-serial loop data from the input loop to bit-parallel for storage in the communication processor's memory, and bit-parallel data from storage to bit-serial data for placement on the output loop. The MLIA ensures data integrity on the loops and controls data exchange with the loop multiplexer(s).

Loop Multiplexer

The loop multiplexer manages data flow between the multiplex loop and associated circuits. Up to eight loop multiplexers may be connected to a single multiplex loop, allowing a maximum of 254 communications lines to be terminated per NPU.

Communication Line Adapter (CLA)

CLAs interface communication lines to the loop multiplexer and provide in-transit single byte buffering; two classes of CLAs, synchronous and asynchronous, are available in several speed ranges:

- Asynchronous interfacing circuits operating at standard speeds of 9600 bits per second or less

- Synchronous character-oriented interfacing circuits operating at 56,000 bits per second

- Synchronous bit-oriented interfacing circuits operating at speeds of 19,200 bits per second or less

The CLAs themselves demand data input and output. The asynchronous CLA supports autospeed detection, therefore a user need not dedicate a CLA for a particular line speed and/or terminal type. When data is being received, a programmed detection feature tests the line speed and code (ASCII, BCD, etc.), facilitating proper handling of the data by the NPU.

MODELS

The 255X Network Processing Unit is available in the following models and options:

Model	Description
2551-1	Memory: Includes 32K words of 475 nanosecond, 16-bit main MOS memory. Throughput Capacity: Nominally rated at 10,000 characters per second at the CDC CYBER channel coupler. Actual throughput is dependent on a number of factors. These include NPU and software, number of circuits connected, block length, circuit activity, circuit speed and available memory. Multiplexer Capacity: Includes loop multiplexers fof interfacing 16 Communications Line Adapters (up to 32 communications lines). Maximum Connectability: Sixteen Communication Line Adapters (CLAs) which define up to 32 communication lines (max), depending on the types of CLAs used. Channel Coupler: The CDC 2558-3, 2558-4 or 10344-1 Channel Couplers provide host computer I/O channel interface for the 2551-1 NPU. The 2551-1 can control up to two channel couplers. At least one 2558-3, 2558-4 or 10334-1 Channel Coupler is required when the 2551-1 is used as a front-end processor. Upgrade: The 2580-3 option is field installable and upgrades the 2551-1 NPU to a 2551-2.

Remote Loading: The 2580-4 option provides the hardware necessary for automatic loading and restarting a remote NPU. This option is field installable, and is required when the 2551-1 is used as a remote NPU.

2551-2 Features of the 2551-2 are the same as those of the 2551-1 except the following:

Multiplexer Capacity: Includes loop multiplexer for interfacing 32 communication line adapters (up to 64 communication lines, depending on the type of CLA).

Maximum Connectability: 127 Communication Line Adapters (CLAs) which define a maximum connectability of up to 254 communication lines. Expansion is attained by adding 2556-10 Expansion Cabinets and 2556-11 Loop Multiplexer Units.

Maximum Main Memory: 128K words (with additional 2554-16/-32 Memory Expansion Units).

Upgrade: None.

Multiplexer Expansion: The 2556-10 Expansion Cabinet and 2556-11 Loop Multiplexer options are used for expansion beyond the basic 32 CLA capacity of the 2551-2 NPU. Each 2556-11 Loop Multiplexer will interface an additional 16 CLAs. A 2556-10 Expansion Cabinet is used to house the additional loop multiplexers. One expansion cabinet is required for every two 2556-11 Loop Multiplexers.

10315 DATA CHANNEL CONVERTER

The CDC 10315 Data Channel Converter allows CDC 3000 series peripheral equipment to be attached to CDC CYBER 170 series computers. The converter enables CDC CYBER 170 computers to perform connect, function, read, write, and status operations on series 3000 peripherals.

DESCRIPTION

The CDC CYBER 170 computer transmits codes to the 10315 converter prior to starting any operation on external equipment. These codes establish conditions in the converter so that the proper 3000-type signals accompany the CDC CYBER 170 computer input/output operations. As many as eight peripheral devices can be connected to the output of the 10315 converter.

MODELS

Model	Description
10315-1	Provides the first data channel converter.
10315-2	This model provides the second data channel converter.

3270/8271 TRANSFER SWITCH SUBSYSTEM

The CDC 3270/8271 Transfer Switch Subsystem provides for switching of peripheral controllers between channels of the same computer system, or between channels of co-located computer systems. The switching operation is controlled manually.

The 3270 Transfer Switch Controller provides cabinetry for mounting 8271 Transfer Switches, and includes power supply necessary to activate switching operation. The 8271 Transfer Switch is a set of relay-operated switches controlled by an externaly mounted alternate-action momentary contact switch.

MODELS

The 3270/8271 Transfer Switch Subsystem consists of the following models:

Model	Description
3270-A	Controls up to four transfer switches
3270-B	Controls up to eight transfer switches
8271-2	Manually operated transfer switch

405 CARD READER

The CDC 405 Card Reader is an input device for use with Control Data computer systems to provide the capability of reading data from punched cards.

HIGHLIGHTS

- Reads 80-column cards at 1200 cards per minute, 51-column cards at 1600 cards per minute
- Dual photo-electric read station permits binary, Hollerith, and ASCII codes
- Pneumatic card handling
- Large (4000-card) hopper and stacker trays

DESCRIPTION

The 405 reader photo-electrically senses data punched in standard 80-column or less cards, reading in a column-by-column fashion. The movement of punched cards is done pneumatically. Cards are picked one at a time from the input tray and proceed to the read station which then reads each card twice and compares the two information blocks to assure accuracy. As directed by the computer, the reader delivers the card to the primary or secondary receiving tray. Cards can be removed from the feed and receiving trays while the reader is in operation. The trays vibrate to provide a constant force at the picker and the stacker regardless of card deck size.

3447 CARD READER CONTROLLER

The CDC 3447 Card Reader Controller is a single channel device, with a full card buffer memory, which provides an interface between the CDC 405 Card Reader and a computer input/output channel. It is contained within the 405 Card Reader cabinet.

HIGHLIGHTS

- Provides buffer memory for one 80-column card image
- Logic to perform most of the controlling and checking functions that would otherwise be performed by the computer
- Performs translation of binary, Hollerith, or ASCII (64-character subset) codes

DESCRIPTION

The 3447 controller acts as an interface between the computer and the 405 reader in the transmission of all data, external function codes, and status codes. The controller transfers data to the computer in 12-bit bytes. The logic of the controller also checks the parity of each code and all data bytes. If an error is detected in a code, the controller sets an error signal and does not act on the code. If an error is detected in a byte, an error signal is generated. The controller is capable of translating binary, Hollerith, or a 64-character subset of ASCII Hollerith card codes to 6-bit internal BCD codes.

The controller contains a buffer memory that holds 80 bytes (one card image). The reader enters data into this memory at the relatively slow rate of 50 milliseconds per card. After a complete card has been read, the 3447 transfers the data (byte-by-byte) at a rapid rate (maximum of 390 microseconds per card). The 405 reader automatically reads another card after the first card has been transferred.

The character set consists of 64 symbols including the English alphabet in uppercase, arabic numerals, punctuation marks, and special symbols.

MODELS

The 3447 controller is available in the following models:

Model	Description
3447	Binary or Hollerith to BCD codes
3447-2	Binary, Hollerith, or ASCII (64-characters) to BCD codes

415 CARD PUNCH

The CDC 415 Card Punch is an output device used with the Control Data computer systems to provide the capability of punching data on cards.

HIGHLIGHTS

- Punches 80-column card at 250 cards per minute using binary, Hollerith or ASCII card codes
- Input hopper capacity of 1200 cards; output hopper capacity of 1500 cards with programmable off-set stacking
- Automatic read after punch

DESCRIPTION

The card punch perforates standard 80-column cards at a rate of 250 cards per minute. Cards are punched row-by-row, using a bank of 80 punch elements. Following the 12-row punch operation, the card is sent to a post-punch read station. The card is read and the number of holes actually punched is verified against the number specified in the punch operation. If in error, the card is identified via the selectable offset feature of the punch.

Input hopper capacity is 1200 cards and the output stacker accommodates up to 1500 cards. Cards can be added to the hopper or removed from the stacker while the punch is in operation.

3446 CARD PUNCH CONTROLLER

The CDC 3446 Card Punch Controller is a single channel device to interconnect the CDC 415 Card Punch and a computer data channel.

HIGHLIGHTS

- Provides buffer memory for one 80-column card image
- Permits translating binary, Hollerith, or ASCII (64-character subset) codes
- Logic to perform controlling, checking, conversion, and formatting functions

DESCRIPTION

The 3446 controller acts as an interface between the computer and the 415 punch for transfer of all data, external function codes, and status codes. It is capable of translating binary, Hollerith, or a 64-character subset of ASCII Hollerith codes.

The controller contains a 12 by 80 turn-around core memory for buffering a card image received from the data channel on a column-by-column basis, beginning with the lower-order column. With the card image in memory, the controller then transfers the data to the card punch on a row-by-row basis.

The character set consists of 64 symbols including the English alphabet in uppercase, arabic numerals, punctuation marks and special symbols.

MODELS

The 3446 controller is available in the following models:

Model	Description
3446	BCD to binary, or Hollerith codes
3446-2	BCD to binary, Hollerith or ASCII (64-character subset) codes

580 TRAIN PRINTER

The CDC 580 Series Train Printers, with Programmable Format Control (PFC), is a complete printer family. This series features 1200, 1600, and 2000 line-per-minute operation, an acoustically-dampened cabinet, enclosed, powered paper stacking, extended-font print capability, and a micro processor-based paper motion control system.

HIGHLIGHTS

- Micro processor-based paper motion control system
- Integrated circuit electronics provide high electrical reliability
- A blower provides constant air circulation to prolong component life
- Paper condition is monitored to preclude possible loss of information
- Pluggable modules simplify removal and replacement of electrical components
- Operates at 1200, 1600, and 2000 lines per minute (model option) with a line width of 136 columns.
- Program selectable six or eight lines per inch (lines per 2.54 centimeters) spacing
- Built-in maintenance features

DESCRIPTION

A printer subsystem consists of a train printer and a single-access controller, packaged together in the same cabinet. A CDC 596 Train Cartridge is required to complete this subsystem.

The controller may be interfaced to CDC CYBER 170, CYBER 70, 6000 or 3000 computer systems. It includes the electronics required to interface a data channel, to control print and paper motion, and through use of memory, to store one line of print and the train image currently being used. This controller also contains a PFC memory which stores the format channel locations as received from the data channel; the PFC memory is operated via external function control.

The character array used by the 580 printer family moves on a horizontal plane and produces uniform, straight-line copy. The type-array cartridge containing 48-, 63-, or 94-character type fonts is easily changed by the operator.

The 580 uses any standard edge-punched, fan-folded paper (up to six copies from a single impression). Forms are advanced under a feedback control system which ensures even line spacing. High throughput is achieved by rapid acceleration to, and deceleration from, maximum skip speed.

Printer design provides easy access to the print-head area, and a minimum of adjustments is required in loading forms. As an additional operator convenience, easy-to-read status indicators are provided on the control panel which alert the operator to any condition requiring attention.

MODELS

The 580 Train Printer is available in the following models:

Model	Speed-lpm	Skip-ips(cmps)	Columns
580-120	1200	70 (177.8)	136
580-160	1600	70 (177.8)	136
580-200	2000	90 (228.6)	136

SPECIFICATIONS

Printing Speed	580-120	580-160	580-200
Nominal (48 characters)	1200 lpm	1600 lpm	2000 lpm
Sustained (24 characters)	1500 lpm	2250 lpm	3000 lpm
Burst (16 characters)	2400 lpm	2850 lpm	3500 lpm

Paper Advance Speed			
Single Space, 6 lpi	12.5 ms	12.5 ms	9.4 ms
Single Space, 8 lpi	12.5 ms	12.5 ms	9.4 ms
Skip Rate, 6 lpi	70 ips (177.8 cmps)	70 ips (177.8 cmps)	90 ips (228.6 cmps)
Skip Rate, 8 lpi	50 ips (127 cmps)	50 ips (127 cmps)	70 ips (177.8 cmps)

Forms Capacity

Internal Stacking:	4 to 20 inches (10.2 to 50.8 cm) wide, 4 to 14 inches (10.2 to 35.6 cm) long
External Stacking:	4 to 20 inches (10.2 to 50.8 cm) wide, 4 to 22 inches (10.2 to 55.9 cm) long
Line Width:	136 columns
Character Pitch, Horizontal:	10 characters per inch (3.9 characters per cm)
Character Pitch, Vertical:	6 or 8 lines per inch (2.4 or 3.2 lines per cm)
Print Cartridge Capacity:	384 characters

677 MAGNETIC TAPE TRANSPORT

The CDC 677 series Magnetic Tape Transports provide data storage with very high performance and heavy duty usage.

HIGHLIGHTS

- Transports feature single capstan, automatic reel hub latching, cartridge load, automatic threading, and power operated window
- Seven-track industry-compatible read or write at 556 or 800 characters per inch (219 or 315 characters per cm) using NRZI recording mode
- Vacuum powered tape scraper and a special tape cleaner that operate during rewind, load, and unload operations

DESCRIPTION

The 677 tape transports record seven tracks of information on magnetic tape. Tape motion is accomplished by a single capstan; vacuum columns act as buffers between the synchronous tape speed and the asynchronous motion of the reel-drive servo system.

The 677 units provide high levels of performance, reliability, and maintainability based on the latest technology. Integrated circuits and modular logic provide ease of maintenance without sacrifice to reliability. Internal diagnostic data loops and extensive unit checking status bits are stored and available for fast diagnosis and corrective action, if required.

MODELS

Characteristics	677-2	677-3	677-4
Tape speed ips (cmps)	100 (254)	150 (381)	200 (508)
Rewind speed in ips (cmps)	60 (152.4)	50 (127)	45 (1143)

679 MAGNETIC TAPE TRANSPORT

The CDC 679 series Magnetic Tape Transports provide data storage with very high performance and heavy duty usage.

HIGHLIGHTS

- Transports feature single capstan, automatic reel hub latching, cartridge load, automatic threading, and power-operated window
- Nine-track recording at 800 characters per inch (315 characters per cm) NRZI and 1600 characters per inch (630 characters per cm) phase-encoded and 6250 characters per inch (2461 characters per cm) group-coded recording (GCR) mode
- Improved throughput via automatic gain control on the GCR models to dynamically adjust for a wide variety of tape qualities
- Vacuum powered tape scraper and a special tape cleaner that operate during rewind, load, and unload operations

DESCRIPTION

The 679 tape transports recor[illegible] cks of information on magnetic tape. Tape motion is accomplished by a single capstan; vacuum columns act as buffers between the synchronous tape speed and the asynchronous motion of the reel-drive servo system.

The 679 units provide high levels of performance, reliability, and maintainability based on the latest technology. Integrated circuits and modular logic provide ease of maintenance without sacrifice to reliability. Internal diagnostic data loops and extensive unit checking status bits are stored and available for fast diagnosis and corrective action, if required.

Advanced recording techniques provide some models with 6250 characters per inch (2461 characters per cm) density, which boosts system throughput, reduces the frequency of input/output errors, and significantly improves error recovery. Additionally, increased density decreases human handling time, reduces physical storage space requirements, and reduces tape procurement costs.

MODELS

Characteristics	679-2	679-3	679-4	679-5	679-6	679-7
Recording density in bits per inch	800/ 1600	800/ 1600	800/ 1600	1600/ 6250	1600/ 6250	1600/ 6250
Recording density in bits per cm	315/ 630	315/ 630	315/ 630	630/ 2461	630/ 2461	630/ 2461
Tape speed (ips)	100	150	200	100	150	200
Tape speed (cmps)	254	381	508	254	381	508
Rewind speed in seconds	60	50	45	60	50	45

7021-3X MAGNETIC TAPE CONTROLLER

The CDC 7021-3X Magnetic Tape Controller provides an interface between the CDC 677 and 679 Magnetic Tape Transports and the CDC CYBER 170, CYBER 70, or 6000 series computer system input/output channels. Up to eight 67X transports of any model type can be intermixed on a 7021-3X controller and will automatically be recognized by model type and addressed accordingly.

HIGHLIGHTS

- Transports are connected to the controller in parallel, making it possible to switch individual drives off-line for maintenance purposes
- State of the art modular design along with the latest in integrated circuitry, results in a subsystem that is highly reliable and easy to maintain
- Comprehensive on-line/in-line diagnostic software is provided
- Built-in maintenance panels and internal data loop checking capabilities are provided

DESCRIPTION

The 7021-3X reads or writes tapes on both the 677 and 679 series of tape transports. Such a subsystem can read or write tapes using industry-standard NRZI, phase encoded, or group coded recording (GCR) methods of recording, depending on tape transport selection. All subsystems can perform character-code translation during data transfer with no degradation of the character transfer rates specified for the various tape transports. Various assembly/disassembly modes of operation for data are also provided. Advanced technology in the GCR tape systems helps boost system throughput, reduces the frequency of errors, and significantly improves error recovery. Error recovery includes two-track, "on-the-fly" error correction in GCR mode and single-track "on-the-fly" error correction in phase encoded mode, assuring a high level of subsystem performance.

MODELS

The 7021-3X is available in the following models:

Model	Description
7021-31	Single microprogrammed control unit, one channel to up to eight tape transports.
7021-32	Dual microprogrammed control units, two channels to up to eight tape transports, two simultaneous data transfers.

885 FIXED MODULE DRIVE

The CDC 885 Fixed Module Drive (FMD) is an electromechanical access storage unit that records and retrieves data on two fixed disk modules. A CDC 7155 Fixed Module Drive Controller is required to interface between the 855 unit and a CDC CYBER 170 series computer system.

HIGHLIGHTS

- Two spindles are included in each unit
- 4.2 billion bits per spindle
- Large block nominal transfer rate of 1.2 million 6-bit characters per second for each spindle
- Disk pack rotational speed of 3600 rpm with an average latency time of 8.3 milliseconds, 16.7 milliseconds maximum
- Average positioning time of 25 milliseconds; 10 milliseconds minimum, 50 milliseconds maximum
- Up to four units (eight spindles) can be connected to create a subystem

DESCRIPTION

The CDC 885 provides access to data organized on two fixed disk modules. Each module has a user storage capacity of 692 million 6-bit characters and uses a sectored addressing method of 644 characters per sector and 32 sectors per track.

The disk module consists of a stack of twelve 14-inch (35.6 cm) diameter, oxide-coated disks. For each module data is accessed by 40 read/write heads mounted on a carriage that is positioned to the required track by a voice-coil linear actuator operating in a closed loop, proportional-servo system.

MODELS

Model	Description
885-11	Single Channel Unit, Bit Serial FMD
885-12	Dual Channel Unit, Bit Serial FMD
885-11F	Single Channel Unit, Bit Serial FMD - Fixed Head
885-12F	Dual Channel Unit, Bit Serial FMD - Fixed Head
885-42	Dual Channel Unit, Four-Head Parallel FMD

7155 FIXED MODULE DRIVE CONTROLLER

The CDC 7155 Fixed Module Drive Controller serves as an interface between the CDC 885 Fixed Module Drive (FMD) and the CDC CYBER 70, CYBER 170, or 6000 series computer systems.

HIGHLIGHTS

- Performs multiple-overlapped seek operations concurrently with one read or write operation to maximize the data throughput rate of the subsystem
- Includes a small programmable processor with a 1K 12-bit data buffer. The programmable capability of the subysystem enables data to be transferred at channel speed for one sector bursts
- Dual controllers in a subsystem provide simultaneous data transfers (read or write or both)
- Provides three types of error recovery: head positioning, address field, and data field
- Controlware supports subsystem function set, including maintenance function set

DESCRIPTION

The basic 7155 controller can control up to four 885 units (eight spindles). The hardware allows for expansion of up to eight 885 drives (16 spindles), eight CDC 844-4X Disk Storage Units, and four channel accesses. The subsystem uses a sectored addressing method of 644 characters per sector and 32 sectors per track, and supports a mix of single-density drives with a capacity of 692 million 6-bit characters and double-density drives each having a capacity of 237 million characters (per spindle). The controller includes a small, programmable processor which enhances the normal operation of the subsystem and provides for more diagnostics and recovery capabilities.

MODELS

The 7155 controller is available in three models. Options may be added to enhance the capacity of the 7155.

Model	Description
7155-1	Connects to one standard input/output channel. Drives up to four 885-xx FMDs (eight spindles) in a mix or match configuration.
7155-2	Connects to one or two standard input/output channels. Drives up to four 885-xx FMDs (eight spindles) in a mix or match configuration.
7155-4	Connects up to four standard input/output channels. Drives up to four 885-xx FMDs (eight spindles) in a mix or match configuration.

Options

10397-1	Adds a second channel operation to 7155-1, adds a third channel operation to 7155-2 (or 7155-1 with one 10397-1 installed) or adds a fourth channel operation to 7155-2 with one 10397-1 (or 7155-1 with two 10397-1s installed).
10398-1	This interface option connects up to eight 844-4x DSUs. Resulting controller drives up to four 885-xx FMDs (eight with 10399-1 option) and up to eight 8444x DSUs in a mix or match configuration.
10399-1	This interface option connects up to four additional 885-xx FMDs. Resulting controller drives up to eight 885-xx FMDs (and eight 844-4x with 10398-1 option) in a mix or match configuration.

MASS STORAGE SUBSYSTEM

The Control Data CYBER Mass Storage Subsystem (MSS) is designed for use with the CDC CYBER 170, CYBER 70 and 6000 Series Computers. The MSS puts total automation in tape storage operation, removes possible operator errors resulting from manual tape-handling operations, and provides on-line storage and a readily accessible tape cartridge library.

The tape cartridges used in the MSS can be handled in the same way disks are handled, without demanding any changes in normal procedures. Also, files are transferred from the MSS to disks without involving the programmer.

The entire data resource is controlled by the MSS operating system. Disk space is eliminated, since only tape tracks required by staged data are assigned. Active files are not restricted to, nor concentrated on, segregated disks. In addition, the MSS permits field expansion by add-on modules and assures that cartridge size is related realistically to file size for easier, more efficient management.

MSS hardware consists of a Model 7880-1 Mass Storage Controller (MSC), a Model 7881-1 Cartridge Storage Unit (CSU), and up to four Model 7882-1 Mass Storage Transports (MST) per CSU, depending on projected storage requirements.

The MSC is housed in two cabinets. One contains a CDC CYBER Mass Storage Coupler (CMSC), and the other contains a Mass

Storage Adapter (MSA). The CSU and MST are housed in complementary cabinets that bolt together.

- MSC

  The Model 7880-1 Mass Storage Controller consists of one CYBER Mass Storage Coupler (CMSC) and one Mass Storage Adapter (MSA). The CMSC allows two CDC CYBER PPU Channels to interface to one MSA on a time-shared basis and has a 4K, 24-bit Data Buffer Memory. The MSA connects to the CMSC and controls up to a total of eight devices (CSUs and MSTs). The MSA is microprogrammed to control the attached devices and contains one data path with 9-track group coded recording (GCR) format capability to exchange data with the MSTs.

- CSU

  The Model 7881-1 Cartridge Storage Unit consists of a storage module with a capacity of 2,000 cartridges, a two-axis selector module and two drawers for entering and removing cartridges. It includes 500 Mass Storage Cartridges. A minimum of two (2) 7882-1 Mass Storage Transports is required with each Cartridge Storage Unit.

- MST

  The Model 7882-1 Mass Storage Transport includes an interstation drive for moving cartridges within the transport, and an automatic tape load/unload movable read/write head for accessing eight data stream pairs, with 129 inches per second (328 cm per second) tape speed and 6250 bytes per inch (2461 bytes per cm) group coded recording (GCR) format capability. Up to four 7882-1 MSTs may be physically connected to a 7881-1 Cartridge Storage Unit.

- Mass Storage Cartridge

  A Mass Storage Cartridge consists of a plastic case containing 2.7 inch (6.9 centimeters) magnetic tape with a usable recording length of 8.3 feet (2.5 meters). The tape is fastened to the case at one end and to the cartridge spool at the other end. Thus, the entire recording length of the tape resides in the MST vacuum columns after a load operation.

  The cartridge case incorporates a removable write-enable pin for file protection. A cartridge with the write enable pin removed cannot be altered by CDC CYBER MSS.

OPTIONS

- MSC 16 Device

  The Model 10390-1 MSA 16 Device Option adds a second eight device interface to the Mass Storage Adapter. This option provides the capability for attaching additional equipments for increased storage capacity and/or for redundancy.

- CSU Dual Path

  The Model 10393-1 CSU Dual Path Option provides an alternate path to the Cartridge Storage Unit. This device provides redundancy by allowing another Mass Storage Adapter to access the CSU if the primary MSA malfunctions.

819 DISK STORAGE UNIT

The CDC 819 Disk Storage Unit is an electromechanical access storage unit that records and retrieves data on disk surfaces.

HIGHLIGHTS

- Phase-lock read recovery
- Absolute addressing
- Carriage offset
- Maintenance aids
- Data strobe offset
- Data transfer rate: 38.7 megabits per second (4 heads parallel) nominal
- Spindle speed: 3600 rpm
- Average access time: 58.33 milliseconds

DESCRIPTION

The 819 is a peripheral mass memory device. It consists of a cabinet containing a disk pack, drive motor, voice coil positioning mechanism, power supply, and logic chassis.

The recording medium consists of 22 magnetic-oxide coated disks (non-removable pack), with 40 data surfaces and one servo surface containing head positioning information. Data is recorded at 6000 bits per inch (2362 bits per centimeter).

MODELS

The 819 is available in the following models:

Model	Capacity	Access Time	Transfer Rate
819-11	2.4 billion bits	58.33 ms	38.7 M bps
819-21	4.8 billion bits	58.33 ms	38.7 M bps

7639 MASS STORAGE CONTROLLER

The CDC 7639 controller provides a data and control path between the CDC 819 disk storage unit and a peripheral processing unit of the CDC CYBER 70, Model 76, a CDC CYBER 170/Model 176 or a CDC CYBER 200 computer system.

HIGHLIGHTS

- Connects and addresses the disk storage unit
- Assembles and disassembles data transferred between the 12-bit peripheral processor channel and 4-head parallel disk storage unit
- Deskews read data from the disk storage unit
- Generates a preamble and sync byte that precedes each data sector
- Generates a 32-bit checkword for each channel

DESCRIPTION

The 7639 controller can control up to eight 819 disk storage units. The basic capacity of an 819/7639 subsystem is 2.4 billion bits, however, it can be increased to 19.2 billion bits with a configuration that includes two controllers and eight disk storage units. Using dual density model 819 disk storage units, the minimum capacity is 4.8 billion bits, which can be increased to 38.4 billion bits with a configuration of two controllers and eight disk units.

The data transfer rate is nominally 3.1 million 12-bit words per second. The maximum possible burst transfer rate is 3.33 million 12-bit words per second.

MODELS

The 7639 controller is available in two models each of which can handle the single or dual density model 819 disk drives:

Models	Description
7639-21	Interfaces one to four disk storage units and permits multiple overlapped seek operations concurrently with one read or write operation.
7639-22	Interfaces to a maximum of eight drives and provides two simultaneous data transfers (read and/or write) to any two drives, while retaining the multiple-seek feature.

CYBER 18/MODEL 20

The CDC CYBER 18/Model 20 is a highly versatile, general-purpose, micro-programmed, 16-bit processor that emulates the CDC 1700 instruction set (basic set plus a new enhanced instruction set).

Execution of systems programs stored in macro main memory is performed under the control of a micro-level program stored in ROM micro memory. The micro level programs also operate input/output channels, service the computer interrupt and program protect systems, and control the operating mode of the processor. They provide additional facility for character and field manipulation, indexing, and other system-oriented processes.

Arithmetic is one's complement, signed, fixed-point, hardware add/subtract/multiply/divide. The arithmetic section consists primarily of several operation registers (I, P, X, A, M, Y and Q) that are interconnected by selectors and interface to the arithmetic and logical unit. The Model 20 has register files available at the microprogram level.

The Model 20 has ten card positions available to support A/Q, ADT or DMA type peripherals. The A/Q channel provides data transfers between Central Processor Unit (CPU) registers and the internal peripheral controllers.

An additional input/output interface is available for the optional operator comment device. The Model 20 also has an integral real-time clock which appears as a CDC 1700 type peripheral to the macro program, and provides a macro level interrupt at a programmable interval.

FEATURES

The Model 20 features include:

- Micro-programmable architecture
- Accommodates 32K through 262K bytes macro main memory
- High reliability and easy maintainability through state-of-the-art technology and advanced diagnostic capability
- Main memory effective read/write cycle time of 750 ns
- Eight addressing modes for accessing main memory
- Main memory word and region protection
- Main memory parity detection
- Priority-oriented interrupt system with 16 levels each of micro and macro interrupts
- Powerful macro instruction repertoire
- Integral real-time clock
- Basic processor supporting a wide range of peripherals
- Modular design CPU and controllers for ease of maintenance
- Automatic program load (deadstart) facility for loader-type peripherals
- High-speed input/output data transfer for integral peripheral controllers
- Input/output communications interface for teletypewriter or RS232-C compatible display terminal
- Optional micro/macro breakpoint controller
- Self test and echo mode tests are included as an aid in trouble-shooting the basic processor and optional controllers. The system is also supported by controlware diagnostics which are included in the Operational Diagnostic System (ODS)

CONFIGURATION

The basic configuration includes processor, cabinet with operator's panel and power supply, and input/output controller for communication console. No main memory is included.

For operation, minimum additional requirements are 32K bytes main memory, an input device such as a card reader, and a comment device such as a conversational display terminal.

MACRO INSTRUCTION REPERTOIRE

The CDC CYBER 18/Model 20 incorporates the basic CDC 1700 instruction set and new enhanced instructions. The repertoire includes one, two and three word instructions. Instruction groups include the following:

- Transfer
- Logical
- Stop
- Shift
- Interrupt
- Parity Generation
- Character/Field Manipulation
- Micro Code Sequence Execution
- Arithmetic
- Jump
- Decision
- Input/Output
- Program Protect
- Loop Control
- Memory Paging Control

PROGRAM PROTECTION

The CDC CYBER 18/Model 20 offers two modes of protection from the damage which may be done by programs accessing memory outside their own region. The traditional word level protection of the CDC 1700 series is featured. This allows individual bytes to be declared protected by setting a bit in memory associated with that byte. A second means of protection is also employed using upper and lower bounds to define an unprotected region. This has the same effect as word protection, except that a large unprotected area can be defined more quickly.

INTERRUPT SYSTEM

The CDC CYBER 18/Model 20 firmware emulates the 16 levels of vectored interrupt featured on the CDC 1700 series computers. This system consists of 15 levels of external interrupt and one internal interrupt.

Certain conditions such as an incorrect instruction, a memory parity error, or a power failure will generate an internal interrupt. External interrupts occur when a computer peripheral device has finished an input/output operation or requires attention.

The strength of the interrupt scheme is its ability to handle a significant number of interrupts in a flexible and efficient manner.

1882 MOS MAIN MEMORY STORAGE

The CDC 1882 MOS Main Memory Storage unit provides read/write MOS memory for a CDC CYBER 18 processor. One protect bit and one parity bit are provided with each two bytes. Memory read/write cycle time is 750 nanoseconds. The 1882-16 adds 32K bytes of MOS memory and the 1882-32 adds 65K bytes of MOS memory. The 1882 occupies one memory position within the CDC CYBER 18 processor unit.

1811-2 OPERATOR CONSOLE

The CDC 1811-2 Operator Console is a self-contained, single station CRT keyboard display terminal. It provides a 1920-character (24 lines of 80 characters) display unit with character-by-character data transmission in either half- or full-duplex modes. The data rate is selectable from 110 to 9600 bits per second. The interface meets RS232-C, CCITT recommendations V24 as applied to asynchronous data communication. The character repertoire includes 128 symbols displayed within a 9 by 7 dot matrix.

1843-1 DUAL-CHANNEL COMMUNICATION LINE ADAPTER

The CDC 1843-1 Dual-Channel Communications Line Adapter (DCCLA) interface allows the CDC CYBER 18 computer series to control synchronous and asynchronous communication lines. The DCCLA allows communication flexibility to interface with other computers, terminals and communication devices. It also provides interface compatability over a wide range of system protocols.

Versatility is achieved by the selectable baud rates, character code lengths (5, 6, 7 and 8 bits) and even, odd, or no-parity operations. Input/output operations are enhanced by the selectable protect feature, allowing executive use of the DCCLA and multiple interrupts. Automatic data transfer provides pseudo-buffering capability which simplfies I/O operations. DCCLA testing is also provided by a diagnostic loop-back feature.

The DCCLA provides two-channel, half- or full-duplex EIA synchronous and asynchronous, with modem control. The communication system may be expanded by adding other 1843-1 DCCLAs.

OPERATION

The DCCLA offers full character buffering on transmit and receive. After being initialized for synchronous operation by the software, the DCCLA inserts and strips off sync characters during normal data transmission.

PROGRAMMABLE FEATURES

The DCCLA can be programmed for parity and synch code characters. Parity is programmable on the basis of even, odd or none. When parity is to be used, parity is inserted and checked by the DCCLA. Sync characters can have up to eight bits in any configuration. In addition, test mode, AQ/ADT I/O mode, and Sync/Async modes are program selectable.

CONFIGURATION

The DCCLA subsystem consists of the 1843-1 printed circuit card with a 20 foot (6.1 m) modem cable. The 1843-1 operates in a minimum system configuration of a CDC CYBER 18 Series Processor, 32K bytes of macro memory, an input device such as an 1829-60 card reader, and a comment device such as an 1811-2 operator console.

OPTIONS

- 50-foot (15.2 m) modem cable
- terminal adapter cable which allows the DCCLA to connect directly to a terminal.

1828-1 CARD READER/LINE PRINTER CONTROLLER

The 1828-1 provides two independent controllers for connection of one card reader and/or one line printer to the processor unit; it occupies one A/Q card position within the processor unit. Card reader controller features are data/control interface between the processor and one card reader; it accepts card reader input data in form of Hollerith code, binary code, or any other desired format; it provides facility for deadstart operation of the processor unit; it performs Hollerith-to-ASCII code conversion during deadstart operation; and it performs normal data transfers under program control. The line printer controller features are data/control interface between the processor and one line printer; it performs data transfers under program control; and it has data buffer facility. An additional feature is the test mode capability of closed-loop operation under software control for diagnostic purposes. Cables are included as part of the card reader or line printer.

1829-60 CARD READER

The CDC 1829-60 Card Reader is a self-contained desk top unit provided with interface control logic and an operator's control/indicator panel. Functional characteristics of the unit are 600 card-per-minute read speed, 1000 card hopper/stacker capacity, 80-column punch card input medium, and a photo-electric read station with light/dark read checking facility. One 7-foot (2.1 m) interface cable is supplied. The 1829-60 operates from 120V ac, 60 or 50 Hz source power. Option 1888-1 is available for 220V ac operation.

1827-60 LINE PRINTER

The CDC 1827-60 Line Printer is a high-speed band printer with internal control logic and an MOS static shift register memory which stores one full line of print data. Sound dampened cabinetry and automatic motor turn-off after 30 seconds without printing contribute to quiet operation.

The print mechanism consists of one heavy-duty hammer per two print columns. Sharing is accomplished by shifting the hammer bank between positions using a servo-controlled voice coil. The print band is easily changed.

The printing ribbon is two inches (5.1 cm) wide by 48 yards (43.9 meters) long. Ribbon velocity is constant in either direction regardless of spool load.

FEATURES

- 600 lines per minute
- 64 ASCII character set
- Quietized cabinet
- 132 columns, ten characters per inch (3.9 characters per cm)
- Six or eight lines per inch (2.4 or 3.2 characters per cm).
- Designed for low grade or recycled paper
- Test print maintenance feature with fault indicators
- 50 or 60 Hz option

The 1827-60 Printer subsystem consists of the printer, a 20-foot (6.1 meter) logic cable, a 15-foot (4.6 meter) power cord, one ribbon and one ASCII 64 character set band.

MAINTENANCE FEATURES

Very little maintenance is necessary for the printer; when required, it is accomplished easily and rapidly. LED indicators on the printed-circuit boards show hammer-driver faults (such as short circuit or excessive power drive time).

Modular design permits field replacement of essential modules for depot repair. The electronics boards swing out from the rear of the print head structure. This provides easy accessibility to both sides of the boards. The printed-circuit boards are connected by flat ribbon cables.

A test print switch in the printer permits off-line checkout of the print mechanism and driver logic.

In addition, diagnostic software, Diagnostic Decision Logic Tables (DDLT) and detailed maintenance procedures make up the total CDC CYBER 18 Operational Diagnostic System (ODS) which provides maximum efficiency in maintaining the system.

1833-1 STORAGE MODULE DRIVE INTERFACE

The CDC 1833-1 Storage Module Drive Interface provides a single CPU A/Q-DMA channel interface to the 1833-3 Storage Module Control Unit. The interface handles all control and status operations via the A/Q channel and all data transfers via the DMA channel. The interface supports the control unit connection of up to eight CDC 1867-xx Storage Module Drives in any mix. Connection to the 1833-3 control unit is via two 25-foot (7.6 m) cable assemblies. The interface occupies one A/Q-DMA position within the processor unit.

1833-3 STORAGE MODULE CONTROL UNIT

The 1833-3 is the control unit for the 1867-10 Storage Module Drive (SMD). It provides control for up to eight drives in any mix of 25 million 8-bit bytes and 50 million 8-bit bytes of formatted data capacity. The control unit handles all SMD data transfers, formatting and error recovery. It provides for either single or dual CPU connection via the 1833-1 SMD interface. The control unit is physically housed in the base cabinet of the first SMD in the subsystem. Input power may be either 50 or 60 Hz, 120 or 220 vac, single-phase.

1867-20/21 STORAGE MODULE DRIVE

The CDC 1867-20 Storage Module Drive is a random-access device using removable CDC 877 Disk Packs (or equivalent) for the storage medium. It has a formatted data capacity of 50 million bytes. The maximum data transfer rate is 1.2 million bytes per second. The average access time is 30 milliseconds. The drive includes the base cabinet, one 10-foot (3.1 m) A cable (daisy chain) and one 20-foot (6.1 m) B cable (star). The CDC 877 Disk Pack is not included. The 1867-20 operates from 120V ac, 60 Hz source power, and the 1867-21 operates from 220V ac, 50 Hz source power.

1860-1/2/3/4 MAGNETIC TAPE SUBSYSTEMS

The CDC 1860-Series Magnetic Tape Subsystems provide reliable, low-cost, and versatile tape storage capability for the CDC CYBER 18 processor. Available subsystems record and read industry standard NRZI 7- and 9-track, 800 bpi (315 bits per cm), 25 ips (63.5 cm per second) tapes. Error correction and detection is accomplished using an LRC and CRC character which is generated and checked by the controller. Single track error correction on 9-track tapes is performed automatically by the controller. Program settable clip levels permit error recovery of marginal data. Up to four tape drives can be connected to the controller, which occupies a single A/Q slot in the CPU chassis.

Normal operating functions of this tape unit include read data, write data, write file mark, backspace, erase, rewind to load point, rewind and unload, recovery read, and controller backspace. The controller also provides a number of self-test modes.

HIGHLIGHTS

- 800 bpi (315 bpcm), 25 ips (63.5 cmps), NRZI recording
- Controller self-test mode
- ANSI interchange compatible
- 7- and 9-track drives, any combination
- Up to four drives per controller
- Single track error correction on 9-track

- Single capstan, vacuum column design
- Digital tachometers
- Programmable recovery clip levels

MODELS

Model	Description
1860-1	Seven-track NRZI tape subsystem consists of a single 1860-72 transport mounted in an 1887-4 Cabinet and uses an 1860-200 (upper) Installation Kit. cable connects from the transport to the 1832-4 controller, which is housed in the CDC CYBER 18 central processor.
1860-2	Seven-track NRZI tape subsystem consists of two 1860-72 transports mounted in an 1887-4 Cabinet and uses 1860-200 (upper) and 1860-201 (lower) Installation Kits. A cable interconnects the two transports inside the cabinet, and a single cable connects from the first transport to the 1832-4 controller.
1860-3	Nine-track NRZI tape subsystem consists of a single 1860-92 transport mounted in an 1887-4 Cabinet and uses an 1860-200 (upper) Installation Kit. A cable connects from the transport to the 1832-4 controller which is housed in the CDC CYBER 18 central processor.
1860-4	Nine-track NRZI tape subsystem consists of two 1860-92 transports mounted in an 1887-4 Cabinet and uses 1860-200 (upper) and 1860-201 (lower) Installation Kits. A cable interconnects the two transports inside the cabinet, and a single cable connects from the first transport to the 1832-4 controller.

EXPANDED SUBSYSTEM

Using the 1887-4 Cabinet, the 1860-200 (upper) Installation Kit and the 1860-201 (lower) Installation Kit, transports may be added to the standard subsystems to provide any combination of 7- and/or 9-track NRZI transports, up to the limit of four per 1832-4 controller.

DIVISION 4

LOOSELY COUPLED NETWORK

SPECIFICATION/DESCRIPTION

DIVISION 4

LOOSELY COUPLED NETWORK

SPECIFICATION/DESCRIPTION

The material contained within this document is subject to change without notice. This document or parts thereof should not be considered a commitment or an offer on the part of Control Data. Commitments will only be made through standard channels and with appropriate management approval. It should be further understood the information contained herein represents a functional intent of an implementation and not necessarily the specific approach used in the implementation. The reader should consider this a planning document.

I. INTRODUCTION

One of the ways large computer systems have been growing in complexity is in the amount of inter-processor communications required within a site. The trend towards distributed functions within a system installation has accelerated these requirements. The direct connection of "co-equal" computers is often desired, in order to provide redundancy or fallback capability. In many cases, the requirement is to connect mainframes of different manufacture, thus compounding the problem even further. These links between processors have traditionally been designed as the need arose and each interface required specially developed hardware and/or software. The development of these interprocessor links has, therefore, been cumbersome and the solutions tend to be inefficient, inflexible, and costly.

With this in mind, Control Data developed a system design strategy that simplifies future system site communication problems. The main motivations for such a design are those of economics, flexibility, ease of implementation, and overall system effectiveness. Additionally, Control Data has developed a strategy that permits gradual implementation steps. Control Data understands that large systems are usually vitally intertwined into the businesses they serve, and become increasingly intolerant of extended interruptions.

The keynotes of an appropriate solution are: (1) to provide as many "common" link elements (i.e., those which provide common functions for as many types of future links as we can envision); (2) to be consistent with the major technological trend toward distributed system functions; and (3) to provide a modular growth path, i.e., one which can be introduced in steps.

II. THE APPROACH

Aside from the philosophical problems of the traditional approach, the detail problems of implementation can be lumped under the broad category of connectibility -- both physical and logical.

Physical connectibility is hampered by maximum length requirements for the various cables; by the number, cost, and bulk of the cables themselves; and by the back panel space needed to attach them. In any composite system, the problems are compounded greatly by the variety of cable types needed. Using the data cable concepts, which were appropriate for stand-alone systems, is a serious handicap when attempting to link processors distributed throughout a large site, just because of these distance constraints.

An even more serious problem is that of "logical connectibility". Matching channel parameters to new peripheral devices as they come along has become a major consumer of software development and maintenance resources.

Similar connectibility problems arise in networks where the links are provided by common carrier facilities. However, it has become quite commonplace to provide simple, logical connections between systems from different makers. In this case, both processors are required to provide two main things; the necessary buffering to accommodate the speed requirements of the communication medium, and the need to conform to a common link control protocol. In addition, if they are to understand each other, a common "language" (such as ASCII) must be used. Having done this, a system can communicate with <u>any</u> other to the extent it is equally capable. The key difference between this philosophical design approach and customary ready-resume channels is that the using system does not pretend to own the communication line or attempt to drive it at his "natural" speed. Therefore, the rules of the Control Data approach are as follows:

Rule 1. - <u>Loose Coupling</u>

The interprocessor communication function is a <u>naturally independent system function</u>, i.e., independent of all user processors. Therefore, its control and implementation <u>should</u> be done independently of the user systems, and should not be thought of as "owned" by any of them.

By adhering to this design philosophy, many of the pitfalls of the traditional tightly-coupled links are avoided. It is not necessary to establish a master for every possible send-receive pair, and only one link control protocol need be designed.

Rule 2. - <u>Bit Serial Transmission</u>

The internode transmission should be bit-serial, with enough bandwidth to accommodate the anticipated peak traffic.

There are many benefits to be realized using a bit serial approach. The main technical reason is to capitalize on the enormous amount of research and know-how that has been and continues to be developed by the communications industry for single-wire systems.

III. THE SOLUTION

Control Data's Loosely Coupled Network (LCN) program is specifically designed to solve the problems of interprocessor and peripheral connectibility as discussed in previous paragraphs. LCN provides an efficient approach to interfacing a variety of mainframes and peripherals in a local network. Control Data's solution to the local networking problem encompasses not only the hardware interconnection media but also comprehensive software and diagnostic capabilities.

The LCN system is comprised of several different hardware components. The transmission medium in the LCN system includes one or more coaxial cable data trunks. These trunks provide reliable communication between attached devices at rates up to 50 million bits per second at 1000 feet. Communications over these trunks at longer distances is possible by reducing the number of attached devices. Each trunk has the capability of supporting multiple drops thus providing for interconnection of many devices in a common network. Programmable Device Controllers (PDCs) are used to interface various devices to the coaxial trunk network via a 50-Mbit Data Set. Figure 1 illustrates the basic LCN interface technique.

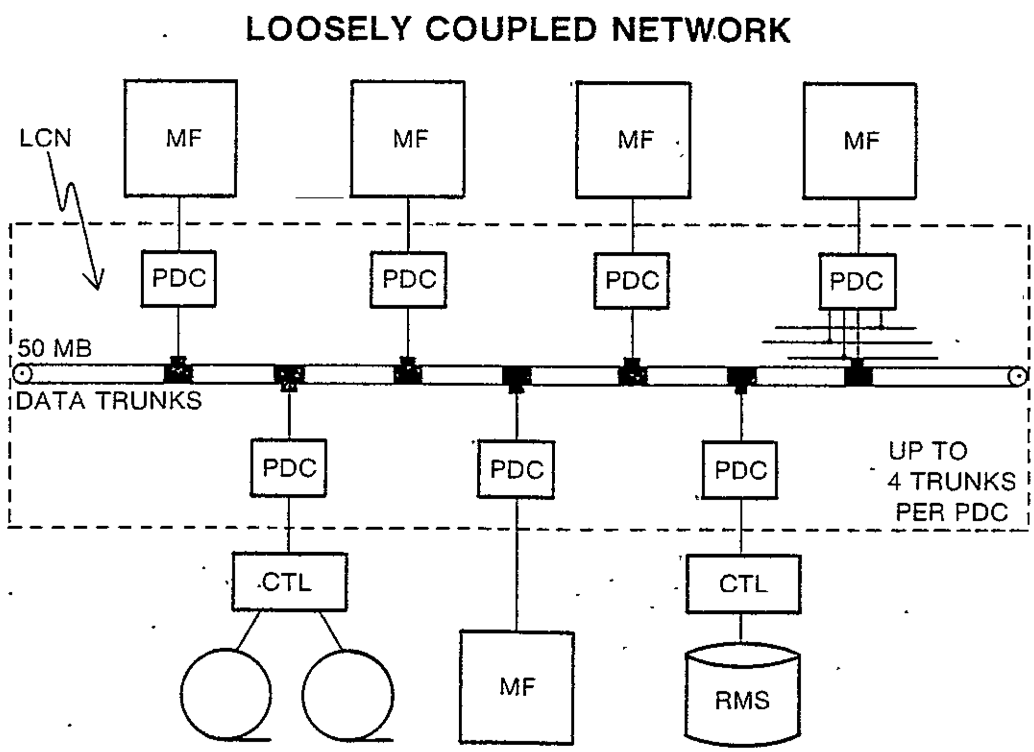


Figure 1. Basic LCN Interface Technique

A PDC can attach to as many as four different trunks. The PDCs are programmable and include buffer memory thus enabling the effective integration of many different devices operating at different rates. Through the LCN, a variety of hardware devices can be linked together.

In addition to the LCN hardware interfaces, Control Data is developing comprehensive software to support various local network applications. Current software developments under NOS are in the areas of mainframe-to-mainframe communication and the sharing of mass storage devices. Other software development is planned for future applications as the LCN program progresses.

The third ingredient in the Control Data LCN offering is in the area of maintenance and diagnostic support. Each CYBER 170 which has an interface into the network will include a set of diagnostics which can be executed in any of the networks PDC's. In addition, other processors connected to the network will contain diagnostics used to assure the integrity of that data link. Each interface into the network will include a set of unique diagnostics which can be exercised in the PDC connected to the various devices. Diagnostics may be initiated in an on-line mode from any Control Data mainframe in the network which is running the NOS Operating System. Additionally, off-line diagnostics may be run through a maintenance interface which connects to a given PDC via an RS232 interface.

Subsequent paragraphs in this document will further define LCN hardware, software, and diagnostic capabilities.

IV. BENEFITS

Control Data believes that the LCN program is a very positive addition to current CDC network offerings. The Control Data approach encompasses a total systems concept which is designed to solve our customers' current and future local networking problems. LCN combines the use of the latest high-speed transmission technology with comprehensive software and diagnostic support to ensure that all facets of a customer's problem are addressed. Major benefits of the Control Data LCN program are as follows:

- CONNECTIBILITY

  LCN enables a customer to readily interface many processors and peripherals in a local network environment. A processor or peripheral connects into the network via one or more PDC. Each PDC has the capability of interfacing with from one to four different trunks. Each trunk can accommodate up to 32 different drops (i.e., 32 different devices assuming one drop per device). In a four-trunk system, up to 128 different devices could be interconnected. Control Data takes advantage of this connectibility via the LCN software now under development. This software will permit multiple mainframes to interact with each other and with peripherals that are part of the local network. Additionally, because LCN is an integral part of the CDC total systems offering, other current interfaces such as remote mainframes, transaction terminals, and interactive terminals can readily be included in the total system.

- CONFIGURABILITY

  LCN offers customers a great deal of latitude in the area of site planning and overall configuration coordination. Customers are no longer constrained by cable lengths and co-location requirements when configuring their systems. The LCN will enable customers to locate processors and peripherals at distances greater than allowed by current channel design.

  The LCN provides a more flexible approach for interconnecting various mainframes and peripherals. In the past, users wishing to link several mainframes together were required to do so via point-to-point connections where each interfaced processor would require an additional piece of hardware. In the case of peripheral devices, multiple hosts were restricted access because of port availability, cable distances, etc. With the advent of LCN, mainframes may tie into a trunk network via one interface box (i.e., a PDC). With one PDC, a mainframe can converse with many other mainframes or various peripheral devices without devoting specific resources to each interface requirement. This type of approach leads to a more simplistic configuration where system operability, control, and maintenance are more readily accomplished.

- MAINTAINABILITY

  The LCN hardware and diagnostic software support are specifically designed to provide the utmost in product and system maintainability. The LCN hardware was designed to meet the highest level of reliability, availability, and maintainability. In addition, the PDC is designed with maximum commonality between different models in order to minimize spares requirements. The combination of comprehensive off-line and on-line diagnostics further enhances the total maintainability of the LCN.

- RELIABILITY

  Reliability, like maintainability, is a key factor in the design of the LCN hardware and software components. Control Data's approach is one of total system reliability--where the combination of reliable hardware and supporting software ensures system processing integrity. Such factors as careful selection of key hardware components, selection of a very efficient protocol scheme, and comprehensive error detection and reporting mechanism all help maximize system reliability. Most of the components used in the LCN sytem are field proven products currently in use in the computer industry. Additionally, LCN provides the capability to configure a system with redundant hardware components (i.e., PDCs and trunks), which alleviate single-point failure problems.

- FLEXIBILITY

  LCN provides customers with an extremely flexible system. The LCN permits ready adaption of not only Control Data interfaces but also other selected vendors' equipments. The PDC was designed such that the majority of components are common, regardless of the device being interfaced. This design provides a modular base that can readily be adapted to interface various equipments as required. With this type of approach, customers no longer have the traumatic interface problems of the past as they integrate new processors and peripherals into their existing configurations. The LCN provides customers with an excellent growth path that allows future expansion as their requirements change.

- SUPPORT

  Control Data's LCN program provides all aspects of support. Because of the system commitment of hardware, software, and diagnostics, Control Data is able to offer a total solution to a customer's local networking requirements and is able to provide a level of support not currently available. In addition, the integration of the LCN program with existing Control Data hardware and software programs provides prospective customers with a comprehensive approach to satisfying a wide variety of system needs.

- PERFORMANCE

  The technology employed in the LCN hardware enables reliable transmission over trunk lines at 50 million bits per second. Maximum trunk utilization is realized due to several key LCN features: (1) Access to the trunk is resolved by a self-synchronizing, rotating priority mechanism. This method guarantees trunk access to all units on the trunk during peak loading. See appendix A for explanation of trunk synchronization. (2) Maximuxm utilization of various system resources is obtained by providing a mechanism whereby data rates of attached devices are buffered from the transmission rates of the LCN. This is accomplished using the buffering scheme employed in the PDC. (3) The protocol used by LCN is designed to provide for a minimum of overhead which in turn permits a maximum utilization of the bandwidth available.

- OPERABILITY

  The LCN improves system operability through the down-line load capability. A PDC can be loaded either through the device interface or trunk interfaces. Therefore, the system operator can initiate a reload should it become necessary to change the system after automatic system initialization.

V. THE PRODUCT

Control Data's LCN strategy encompasses a wide variety of different device connections and various software offerings. Figure 2 illustrates the various hardware interfaces that are either under development or being considered for development.

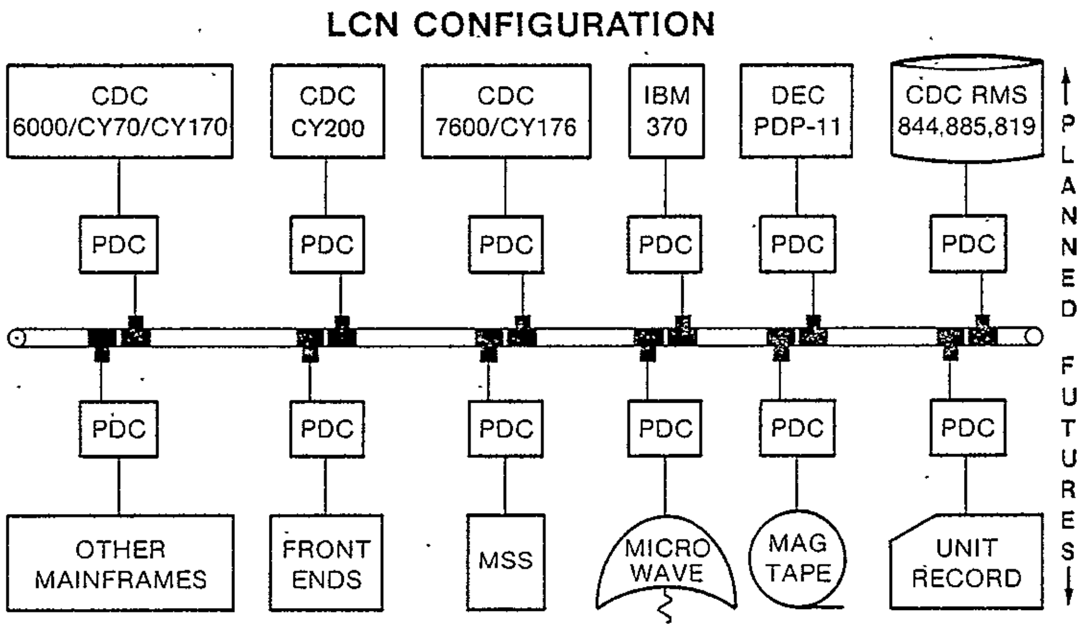


Figure 2. LCN Interface Types

The various interfaces shown above the trunk are currently under development. The interfaces below the trunk are currently being analyzed for future availability. The flexibility of the LCN design will permit future program additions with a minimum of special effort.

Software support for the various LCN interfaces is being done under the NOS Operating System for the 6000/CYBER 70/CYBER 170 line and under the CYBER 200 Operating System for the CYBER 200 System.

NOS software currently being developed includes a host-to-host link for the staging of permanent files, input files, output files, and application-to-application interaction. In addition, other NOS software will be implemented as additional LCN interfaces are identified. The CYBER 200 system includes LCN as the basic input-output mechanism and supported with standard software. Included in this area are support of 819 disks, a CYBER 18 Maintenance Control Unit, and 6000/CYBER 70/CYBER 170 station support .

Support for non-Control Data LCN interfaces will be at the PDC level. The PDC models that interface non-CDC mainframes will include the appropriate device interface and microprogramming logic necessary to easily adapt these equipments into the LCN configuration. Additionally, the host-to-host software being developed for the Control Data mainframes adheres to a high-level protocol that can be readily adapted in non-NOS Operating System environments. Control Data will publish sets of specifications that define interface requirements for non-CDC mainframes.

I. LCN HARDWARE COMPONENTS

The Programmable Device Controller (PDC) is the principal element of a generalized I/O system that uses high-speed serial channels as interconnecting data paths between multiple processors and peripheral equipments. The generalized system, referred to as the Loosely Coupled Network (LCN), has been under development by Control Data for several years. The control of the serial channel is distributed to all resident PDCs rather than being centralized in one network control processor.

Figure 3 shows a block diagram of a PDC. The PDC consists of five functional parts, four of which are common and one of which is unique to the device or channel being interfaced. The trunk interface consists of hardware and microcode that matches the PDC both electrically and logically to a high-speed (50 megabits per second) serial trunk. The trunk interface is in turn comprised of two sets of logic, a trunk control unit interface (TCI) and from one through four trunk control units (TCUs). The number of TCUs that would be required depends on the number of different trunk lines present. The Data Set provides the modulation/demodulation associated with the transmission of data over the coaxial cable. The memory is used for temporary data storage, allowing various devices with differing data rates to use the trunk. The memory also retains the executable instructions called Controlware. The processor controls the PDC resources and manages data flow. The device interface is a unique set of hardware that matches the device or processor channel to the PDC internal bus. The PDC design is such that common parts can be used with a multitude of unique processors or peripheral equipments. This commonality minimizes unique part types, reduces new unit design time, and most importantly assures a controlled serial trunk system structure. The maintenance interface provides a means of connecting an external load device as well as a maintenance control terminal.

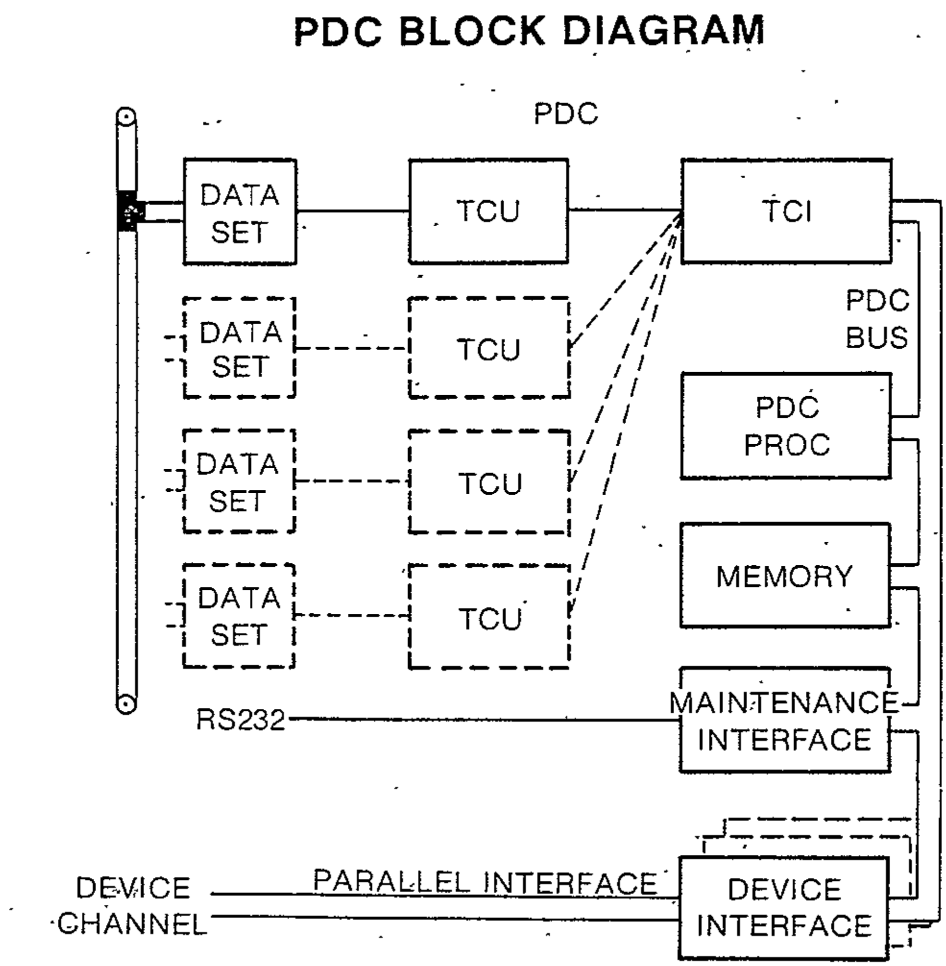


Figure 3. Block Diagram of PDC

- PDC INTERNAL BUS

  The PDC internal bus is used for inter-element communications. The bus consists of 16 data bits, two parity bits, 16 address bits, and several control bits. Bus usage is allocated equally among three elements: the trunk interface, the processor, and the device interface. Each element has a time slice approximately 106 nanoseconds wide which occurs once every 320 nanoseconds. Time slice allocation allows all three elements to access the bus, and therefore the memory, at a guaranteed 50-Mbps rate (16 data bits every 320 nanoseconds).

- PDC MEMORY

  The PDC memory isolates the synchronous serial trunk and asynchronous connected devices, while buffering data rate differences between them. The size of the memory depends upon the attached device and the particular applications. A maximum memory configuration is 65K of 16-bit words. The cycle time of the PDC memory is 106 nanoseconds.

- PDC PROCESSOR

  The PDC contains a hardware processor and controlware microcode which manage the PDC resources and execute system functions. The processor under controlware instructions normally performs the following tasks:

  - Managing PDC resources such as allocation of buffer memory and reservation of the PDC for a particular message source.

  - Handling message flow which includes generating message headers, monitoring data transfers, and interpreting received messages.

  - Initiating error recovery procedures on serial channel transmission errors as well as device errors.

  - Executing various system functions which may include queuing processes, executing I/O processes through the serial channel and device interfaces, handling device drivers, translating message formats if required, generating autoload messages if required.

The PDC has a 16-bit processor constructed from 4-bit microprocessor chips that are microcoded to yield a processor instruction set. The microcode runs at a cycle time of 106 nanoseconds. The number of microcode references per processor instruction varies depending upon the instruction being executed; however, the average is approximately six to eight microcode references. This gives an average processor instruction time of 960 to 1280 nanoseconds.

- MAINTENANCE INTERFACE

  The serial RS232C Maintenance Interface provides an alternate means of loading the PDC and controlling off-line diagnostics. With an optional terminal and suitable load device the maintenance personnel can enter diagnostics directly into memory, read out portions of memory or enter instructions for problem isolation.

- DEVICE INTERFACE

  The device interface electrically adapts a device channel to a PDC. Other key functions that may be performed by the device interface include:

  - Transfer of data and commands between an attached device and the PDC.
  - Assembly/Disassembly to handle different word sizes.
  - Device control for passive devices like disk and tape.

  The device interface is that set of logic which is unique to a PDC, depending on the device being interfaced. Currently scheduled device interfaces include CYBER 170, CYBER 200, IBM 370 MUX channel, DEC PDP UNIBUS, 844/FMD, CYBER 18, and 819. Other device interfaces are currently being analyzed and will be developed as appropriate. With the exception of the DEC 11, all the device interfaces include a single channel connection between the PDC and the device. The DEC 11 PDC will contain a single channel interface as standard with options to connect up to four DEC 11s via a single PDC.

- TRUNK INTERFACE

  The trunk interface performs the following functions:

  - Interface the data set.
  - Add/Delete the serial trunk protocol envelope which includes Cyclic Redundancy Code (CRC) generation and detection.

- Interpret message functions and react accordingly.
- Generate response messages to ensure closure for all valid incoming messages (some form of response will always be generated even if the PDC processor or attached device is unavailable).
- Accesses the trunk if the PDC needs to send a message.

The trunk interface can operate in two modes: message mode and streaming mode. Message mode is the normal mode of operation in which all transactions consist of a command and response message pair. See appendix B for explanation of the message formats. At the end of the message pair, control of the serial trunk is relinquished to allow other PDCs on the trunk to use the trunk. Streaming mode is used in special cases where a high rate is required. In streaming mode, the trunk is held between message/response pairs, allowing the next message to be sent as soon as it is ready.

Access for trunk usage is resolved by a rotating priority mechanism. This method guarantees trunk access to all units on the trunk during peak loading of message mode traffic.

The standard trunk interface is comprised of a Trunk Control Unit Interface (TCI) and a Trunk Control Unit (TCU). Additional TCUs can be field installed to permit access to up to four different trunks.

- SERIAL TRUNK HARDWARE

A 50-megabit data set and coaxial transmission medium are used for the serial trunk. The data set uses a phase-modulated carrier system to transmit data in a synchronous burst mode. High quality coax cable and type-F connectors are used to eliminate any possible ground and EMI/RFI problems.

The performance objectives for the data set and transmission medium are:

- 50-Mbit transmission rate.
- Up to 32 data set attachments per serial trunk.
- Trunk length maximum of 1000 feet with 16 attachments.
- Longer trunk lengths (greater than 1000 feet) are possible with fewer attachments and/or higher quality cable.

A 16-bit Cyclic Redundancy Code (CRC) is included in every message frame transmitted across the serial trunk. The CRC is an extremely powerful error detection mechanism in that single- and multiple-bit errors anywhere in the message frame are detected.

II. LCN SOFTWARE

Control Data is currently developing software support for various LCN applications. The following application areas are under development:

- Host-to-Host Communications (NOS Operating System)
- Shared RMS (NOS Operating System)
- CYBER 200 System Software

In addition to the current LCN software activity, Control Data is investigating support for various other LCN applications. Included in this candidate list are:

- 6250 BPI Tape System
- Mass Storage System
- Communication Front Ends
- Micro-Wave
- High Performance Peripherals

- Host-to-Host Communication

  Control Data is currently developing host-to-host software that will enable multiple mainframes (design maximum is 32) to interface with each other via LCN. See appendix C for proposed second-level channel protocol. Figure 4 contains a block diagram showing a representative configuration for a host-to-host LCN environment. This host-to-host software will provide the following capabilities:

  - Transfer of permanent files
  - Transfer of queue files (jobs and output files)
  - Application-to-application interaction
  - Remote host connection

HOST TO HOST CONFIGURATION

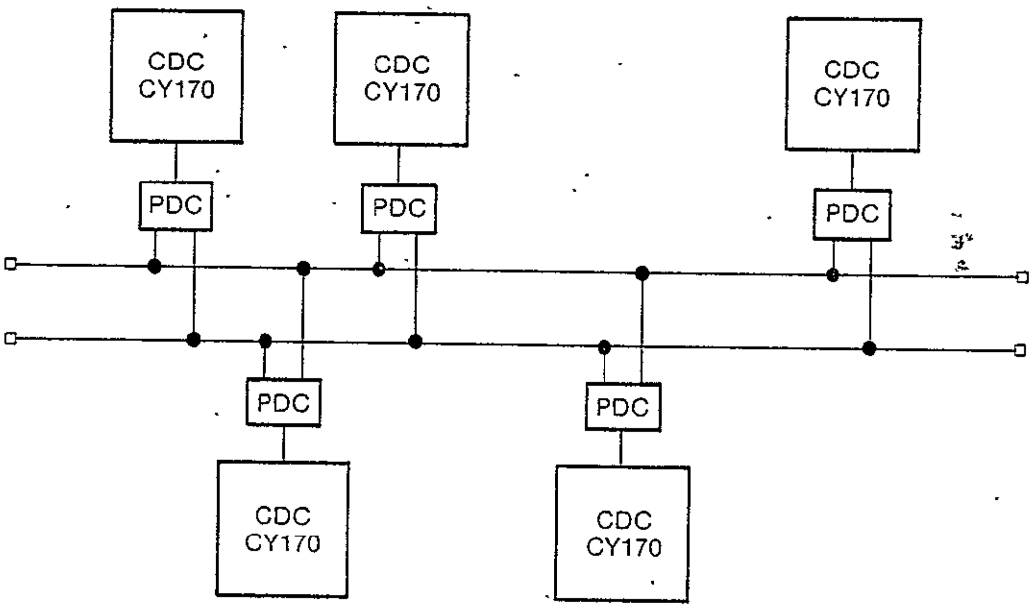


Figure 4. Host to Host LCN

The LCN host-to-host software is being implemented under the NOS Operating System and under CYBER 200 operating system. The base for this software is under NOS the Network Access Method (NAM) module which is an integral part of Control Data's comprehensive Network Host Products (NHP) capability. This type of implementation approach ensures the development of a coordinated and cohesive networking capability that encompasses local networks as well as communication networks.

The Control Data NHP system is very modular in nature and provides an excellent base adapting additional interface requirements. The host-to-host software being developed is referred to as Remote Host Facility (RHF) and is an independent module that will direct user LCN requests to NAM for processing. Control Data's NHP system uses extremely flexible and adaptable protocols which are designed to provide transparent interaction into the network. The adoption of this approach also provides an excellent base for the interfacing of other mainframes (either Control Data or not) which do not run the NOS Operating System.

- Shared RMS

Control Data is developing software that will enable multiple CDC mainframes operating with NOS access to common RMS. This software will initially support 844 and 885 disk subsystems under control of the NOS Operating System. The 819 disk subsystem is only supported by the CYBER 200 Operating System.

The LCN shared disk software will be done in two phases. Figure 5 and 6 shows a configuraton diagram for Phase I and II respectively. Phase I will provide for sharing of RMS by up to eight NOS mainframes. In this phase, all mainframes connect directly to the disk subsystem and the LCN may be used for communication, coordination and recovery activities. Phase II implementation includes the attachment of RMS directly on the LCN trunk.

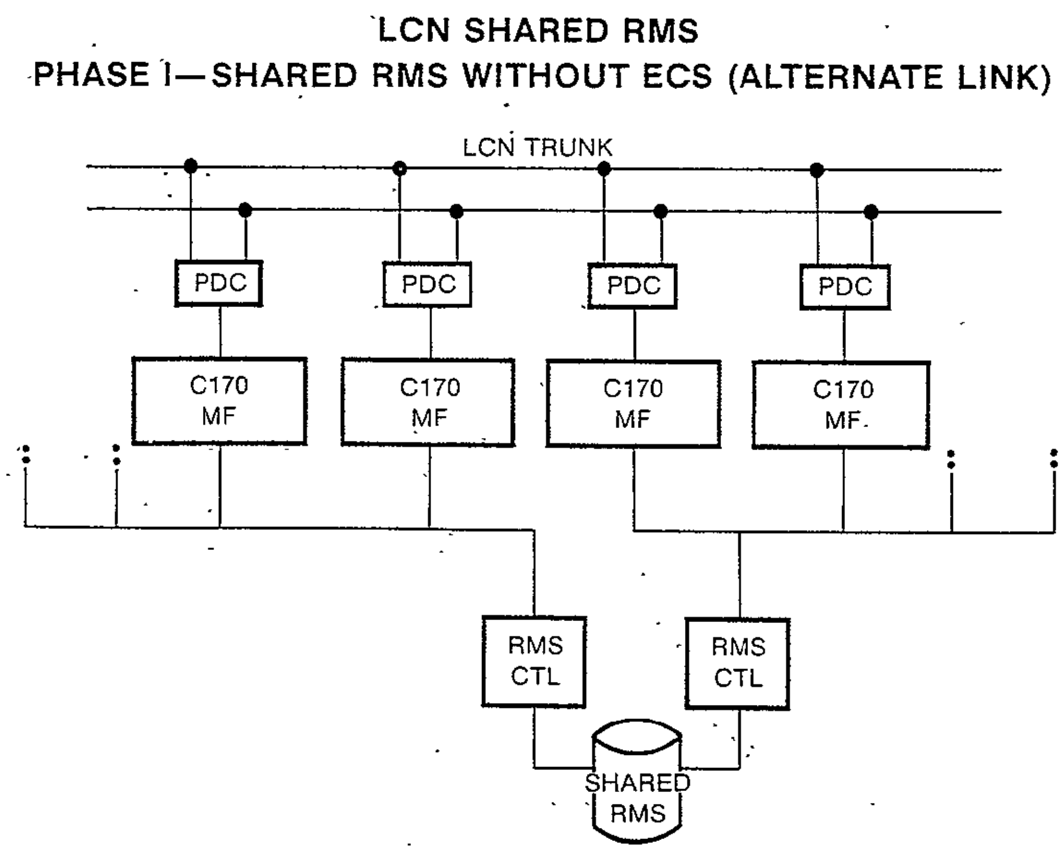


Figure 5. Phase I - Shared RMS Without ECS

LCN SHARED RMS
PHASE II - SHARED RMS WITH PDC I/O

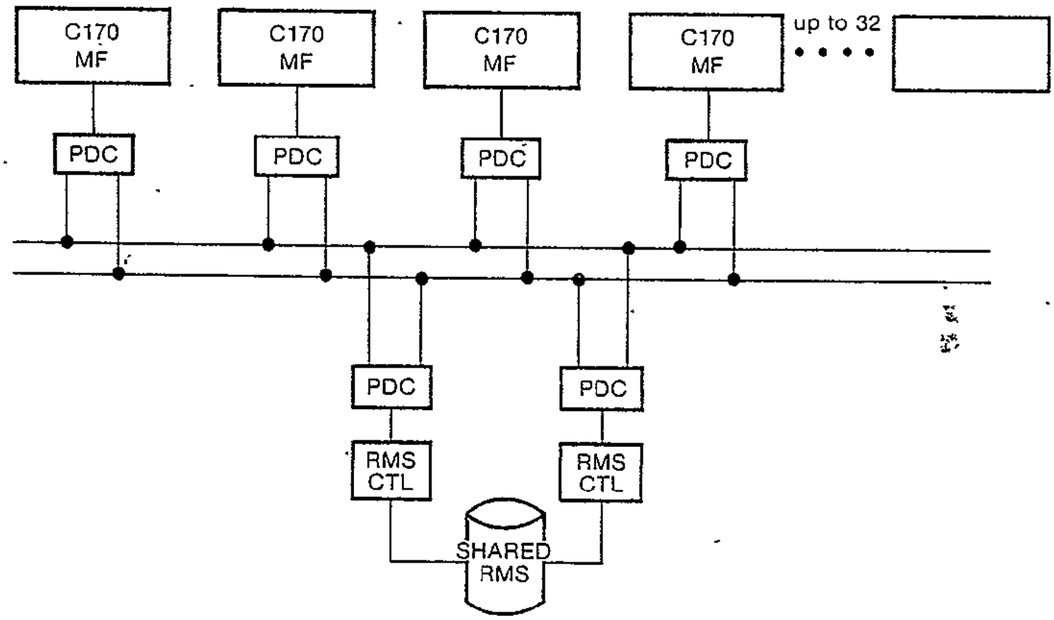


Figure 6. Phase II - Shared RMS With PDC I/O

- CYBER 200

The CYBER 200 I/O system uses a CYBER 170 Computer System, running under control of NOS operating system, to interface all peripheral equipment except mass storage. Mass storage is connected via the Loosely Coupled Network (LCN) directly to the CYBER 200. All communication and data transfer between CYBER 200 and the CYBER station and between the CYBER 200 and its mass storage system takes place via LCN. The CYBER 200, CYBER Station, CYBER 18 Maintenance Control Unit (MCU) and 7639 mass storage controller interface to the trunk through a Programmable Device Controller (PDC). The types of PDC interfaces used in the CYBER 200 system are as follows:

CYBER 200 interface
CYBER 18 interface
CYBER Station interface
7639 interface

The CYBER 200 PDC reads and writes CYBER 200 memory in response to messages directed to it from the trunk.

The CYBER 18 PDC is used to down-line load other PDC's on the trunk and to collect maintenance data.

The CYBER Station PDC interfaces the CYBER 170 System into the LCN in order to provide various station functions for the CYBER 200. Some of the key station functions include job entry, output disposition, tape and disk file processes, and operator interaction.

The 7639 PDC contains the driver for the 7639 storage system. It accepts Read/Write commands at the higher CYBER 200 message level, queues and dequeues them, performs address translation (block number to head, sector and position), and functions the controller to transfer the data to and from its buffers. It also performs error recovery on the transfers and sends error logs on the channel for eventual logging on a CYBER 18 file. Normally due to the high data transfer rate of the 819 and the desire to maintain uninterrupted data flow, a CYBER 200 system will be configured with one dedicated trunk per 7639/819 combination. A typical configuration for the CYBER 200 is shown in Figure 7.

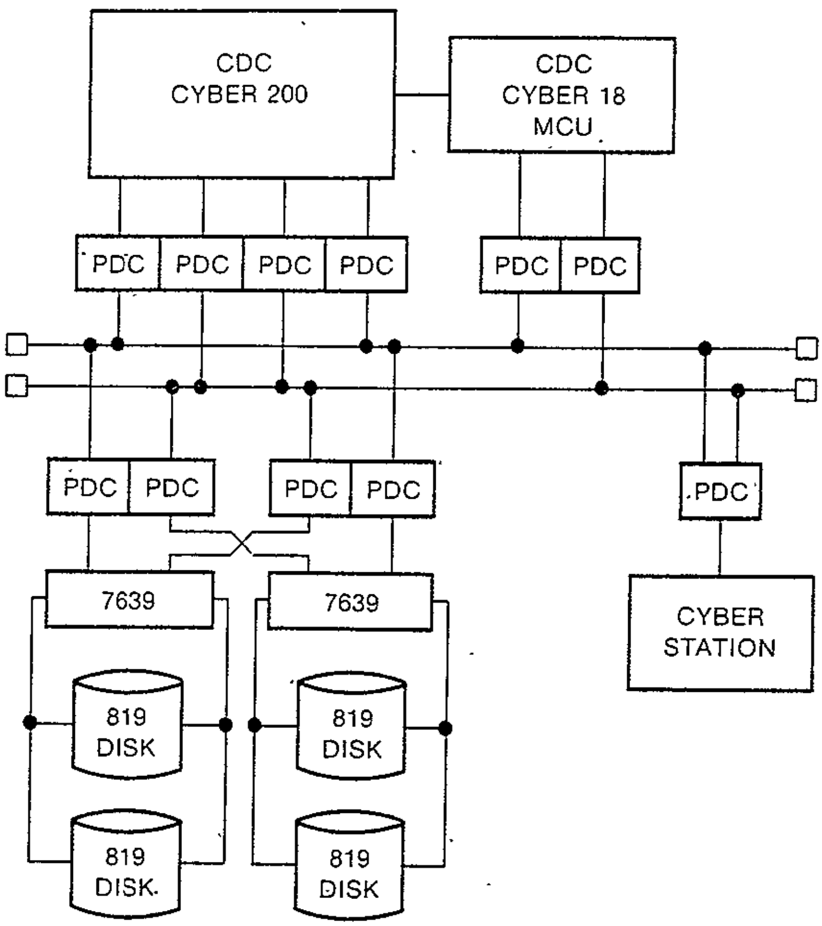


Figure 7. Typical CYBER 200 System Configuration

III. LCN MAINTENANCE/DIAGNOSTICS

Comprehensive, maintenance software will be provided with standard Control Data LCN systems. Where feasible, maintenance of PDC's is performed in conjunction with Control Data's concurrent maintenance capability and uses the Hardware Performance Analyzer (HPA) facility.

Concurrent Maintenance is the diagnosis, repair, and testing of a subsystem or device while the system is performing its normal function. Diagnostics run in parallel with the operating system as normal user jobs are running.

The Hardware Performance Analyzer is a program operating under NOS that tracks equipment usage statistics as logged by the operating system. HPA maintains a listing of equipment errors and calculates the error-rate value. The error rate is compared with a pre-defined standard of expected performance for the equipment. When equipment performance falls below expected standards, a maintenance alert message calls for remedial action.

PDC diagnostics can be loaded either from a local CYBER mainframe or down-line from a CYBER attached to the LCN Trunk.

Maintenance Software is intended to provide the following functional level of LCN integrity:

- Power-on and load-time confidence tests
- First-level PDC diagnostics
- On-Line system diagnostics
- Stand-alone CPU diagnostics
- On-Line Network type diagnostics
- System CPU error logging of LCN failure data

Power-On and Load-Time Confidence Tests (PDC)

These tests are initiated at the completion of a successful power on sequence. They also may be operator initiated or callable when desired from the CPU. This test verifies the operations of the PDC using a building block testing philosophy. The verification test indicates successful completion or error halt. The error halt readout contains codes that allow the operator to determine if the problem is operator correctable or requires a service call.

First Level PDC Diagnostics

PDC diagnostics consist of three segments:

- Utility tests
- Linked PDC diagnostics - First level
- Additional callable tests

On-Line System Diagnostics

These diagnostics run as a problem program concurrent under the operating system. The diagnostic will test all commands and functions. The tests will operate in a most-simple to most-complex mode which allows some degree of isolation. All subsystem failure status and associated sense bytes are provided to the system level. Since these diagnostics run under the operating system, they are also able to initiate execution of the PDC diagnostics. The hardware supports loop back tests from the CPU level.

Stand alone CPU Diagnostics

These diagnostics are similar to the on-line diagnostics but have the additional capability of executing commands that the operating system will not allow. Also, special I/O capabilities outside of standard operating system mode would be available. These tests can also be used to determine if the problem is in the host operating system or the network.

Utility tests

The operator initiates these tests for purposes of verifying trunk and device interface integrity. These tests have the capability of being selectable for one of several test patterns.

Linked PDC Diagnostics

These tests are the prime fault detection and isolation diagnostics for troubleshooting. They are normally run in a predesignated order but can be individually callable and looped upon as required. Error codes derived from these tests will determine what further tests should be run or which logic modules are required to be replaced.

First-level diagnostics diagnose to a logic module level. The diagnostics are capable of checking all functions of the PDC not requiring CPU action which includes the following:

- Ability of the TCU and DI to handle worst-case conditions.
- Memory operation which includes Controlware and buffer memory space.
- Maintenance interface operation

Additional Callable Tests

These are callable in-depth troubleshooting tests restricted to operation from the maintenance panel for customer engineers' use.

These tests also have capability of running real-time tests to determine network degradation and promote predictive maintenance.

Controlware In-Line Maintenance Software

The controlware provides an error record for any PDC error recovery. These records are stored in a subsystem file and presented to the system. The record contains the sense information, number of retries, and whether or not the recovery was successful.

System In-Line Maintenance Software

System error recovery and logging for the PDC is accomplished by the operating system.

APPENDIX A
PERFORMANCE OF TRUNK ALLOCATION CONTENTION ELIMINATION (TRACE) METHOD

SCOPE

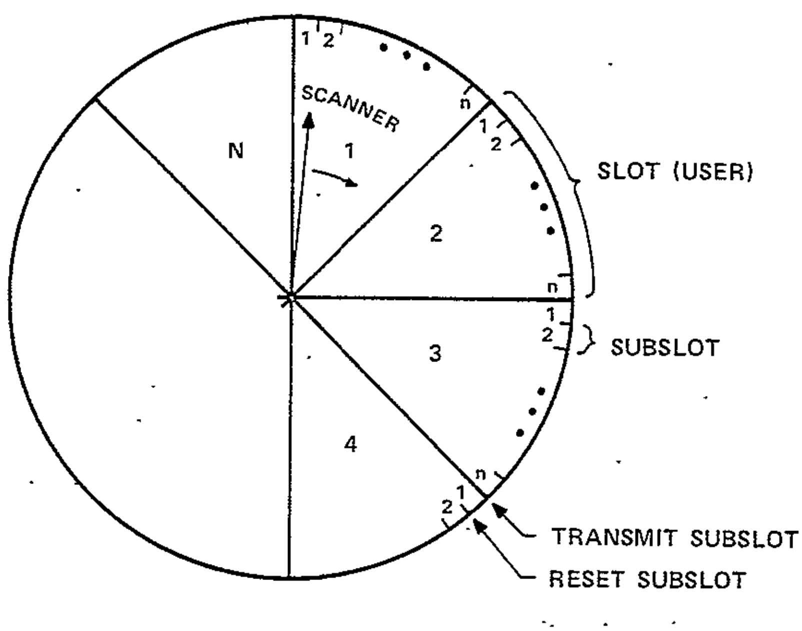
For several years RADL has been experimenting with a time-shared 50 MHz serial coaxial trunk intercommunication subsystem. Such a system offers many advantages over older, more conventional channel communication methods. However, the fact that the trunk is time-shared between a multiplicity of users implies the possibility of contention -- the simultaneous transmission of two or more messages by different users resulting in garbled reception by the intended receivers.

Techniques to prevent contention while still sharing the trunk have recently been proposed. This report describes the methods and assumptions used to simulate the performance of a trunk using the Trunk Allocation Contention Elimination (TRACE) system.

Two different simulation languages were used and simulations on each were run to verify each other. The two simulation languages used were GPSS and ASPOL.

TRACE METHOD

The TRACE method assigns time intervals to each user on the trunk by means of a hardware scanner which is divided into slots and subslots. See figure below. There must be at least one slot for each user but more slots are permissible. Subslots divide slots into convenient smaller time intervals. One slot time must be greater than twice the total transmission line propagation delay. Each user is assigned a unique slot number (user number). Each user has his own scanner which is kept in approximate synchronism with the scanners of all other users. A complete revolution of the scanner is called a cycle. Each user maintains a cycle counter.



BASIC RULES

- A user may transmit a command only at the beginning of his slot time. This eliminates contention since each slot is long enough that all users will see the trunk go active during the transmitter's slot so no other user will begin to transmit.

- If a user has no request to transmit at his slot time he increments his cycle counter. If his cycle counter reaches a limit value the user must transmit a resync command. In this way the various users are kept in approximate synchronization even when there is no message traffic.

- When a user detects a transmission on the trunk he must:

  a) stop his scanner;

  b) monitor transmission to determine who sent it and who it is for;

  c) reset his cycle count to 1;

  d) if transmission is okay, set slot number = sender*;

  e) set subslot number to 2 (reset)**;

  f) wait for end of transmission;

  g) if transmission for this user, send response;

  h) wait twice total line delay for any further activity;

  i) if further activity go to f;

  j) if not, Start scanner.

- A few special rules are needed for system startup and error recovery.

---

*Other special rules could be implemented for fixed priority, multiple slots, etc., if desired.

**Steps c, d, and e accomplish resync for each user.

APPENDIX B
LCN CHANNEL PROTOCOL

B1.0 INTRODUCTION

The purpose of defining the Loosely Coupled Network Channel Protocol (LCNCP) is to simplify the present PDC protocol while retaining its key features. Less hardware and software should be required to implement this protocol.

The LCNCP is a byte-oriented, code independent, modular data link (trunk) control protocol. It is designed for a conference multipoint interconnect and specifically allows multiple active channel connections. The LCNCP protocol is not compatible with any other serial channel protocol.

This document defines in detail the frame structure used in all LCNCP transmissions. It describes the structure, formatting, and significance of the various fields in the frame as well as frame delimiters and frame check sequences.

B2.0 MESSAGE FRAME STRUCTURE

B2.1 General

The vehicle for all command and response information is called a message. Each transmission on the trunk consists of only one message frame. In all but a few special cases, communication between two elements X and Y consists of a pair of message transmissions: a command message transmitted from X to Y and a response message transmitted from Y to X. The command and response message frame structure is shown below.

B2.1.1 Command Message Frame Structure

A valid command message is a minimum of 10 bytes in length following the frame synchronization sequence and must conform to the following structure:

P, F, T, FUN, A1, A2, RP, S, L1, L2, FC1, FC2, I, FC3, FC4

where:

P	=	preamble of all ones preceding sync frame
F	=	message frame synchronization byte
T	=	destination address field
FUN	=	function field
A1,A2	=	access code field
RP	=	resync parameter
S	=	source address field
L1,L2	=	length field
FC1,FC2	=	header frame check sequence field
I	=	information field
FC3,FC4	=	information frame check sequence field

Frames containing only link control sequences form a special case where no I field is present.

The command message frame structure is illustrated in figure B-1. Each element of the frame is detailed in section B2.2.

```
                     ----------------------------------------
                     ^                                |    |
                     |          Preamble              |    |
                     \                                |    |
           Variable      Minimum of 16                | P  |
                     \                                |    |
                     |          One Bits              |    |
                     v                                |    |
                     ---------------------------------|----|
                     ^                                |    |
                 8   |           Frame                |    |
                 Bits|      Synchronization           | F  |
                     |           Byte                 |    |
                     v                                |    |
      -----          ---------------------------------|----|
        ^        8   ^                                |    |
        |        Bits|    Destination Address         | T  |
        |            v                                |    |
        |            ---------------------------------|----|
        |        8   ^                                |    |
        |        Bits|       Function Field           |FUN |
        |            v                                |    |
        |            ---------------------------------|----|
        |        8   ^                                |A1  |
        |        Bits|        Access Code           --|----|
        |            v                                |A2  |
Span of |            ---------------------------------|----|
header  |        8   ^                                |    |
FCS     |        Bits|     Resync Parameter           |RP  |
check   |            v                                |    |
        |            ---------------------------------|----|
        |        8   ^                                |    |
        |        Bits|       Source Address           | S  |
        |            v                                |    |
        |            ---------------------------------|----|
        |       16   ^                                |L1  |
        |        Bits|      I Length Field          --|----|
        |            v                                |L2  |
        |            ---------------------------------|----|
        |       16   ^                                |FC1 |
        |        Bits|       Header Frame           --|----|
        v            v  Check Sequence Field          |FC2 |
      -----          ---------------------------------|----|
        ^            ^                                |    |
        |            |                                |    |
        \            \   Information Field            | I  |
      Span   Variable                                 |    |
       of            \                                |    |
  Information        |                                |    |
       FCS           v                                |    |
      Check          ---------------------------------|----|
        \       16   ^                                |FC3 |
        |        Bits|   Information Frame           -|----|
        v            v  Check Sequence Field          |FC4 |
      -----          ----------------------------------------
```

Figure B-1 - Command Message Structure

B2.1.2 Response Message Frame Structure

A valid response message is a minimum of 10 bytes in length following the frame synchronization sequence and must conform to the following structure:

P, F, T, FUN, P1, P2, P3, S, L1, L2, FC1, FC2, I, FC3, FC4

where:

P1 = not used

P2 = TCU/TCI status

P3 = not used

and all other elements are identical to the command frame elements.

The response frame structure is illustrated in figure B-2. Each element of the frame is detailed in section B2.2.

B2.2 Frame Elements

B2.2.1 Frame Synchronization (P, F)

All messages open with a frame synchronization sequence. The sequence starts with 16 or more one bits and ends with a synchronization character (binary configuration 01111110). The frame synchronization sequence serves as a position reference for all characters in the message, and initiates transmission error checking. At the start of a data transmission a minimum of 16 one bits must be sent before the sync character is sent. Two equipments that are performing a streaming operation can send many one bits at the start of a transmission to hold the serial channel until a message is ready to be sent.

NOTE: No bit insertion/deletion (such as IBM's SDLC protocol) is used in the LCN protocol.

```
                   --------------------------------
                   ^                          |   |
                   |        Preamble          |   |
                   \                          |   |
           Variable    Minimum of 16          | P |
                   \                          |   |
                   |        One Bits          |   |
                   v                          |   |
                   --------------------------------
                   ^                          |   |
              8    |         Frame            |   |
              Bits |    Synchronization       | F |
                   |         Byte             |   |
                   v                          |   |
     -----         --------------------------------
     ^        8    ^                          |   |
     |        Bits |   Destination Address    | T |
     |             v                          |   |
     |             --------------------------------
     |        8    ^                          |   |
     |        Bits |   Response Function      |FUN|
     |             v                          |   |
     |             --------------------------------
     |        8    ^                          |P1 |
     |        Bits |      Parameter 1       --|---|
     |             v                          |P2 |
Span of |          --------------------------------
header  |     8    ^                          |   |
FCS     |     Bits |      Parameter 2         |P3 |
check   |          v                          |   |
     |             --------------------------------
     |        8    ^                          |   |
     |        Bits |      Parameter 3         | S |
     |             v                          |   |
     |             --------------------------------
     |        16   ^                          |L1 |
     |        Bits |    I Length Field      --|---|
     |             v                          |L2 |
     |             --------------------------------
     |        16   ^                          |FC1|
     |        Bits |     Header Frame       --|---|
     v             v  Check Sequence Field    |FC2|
    -----          --------------------------------
     ^             ^                          |   |
     |             |                          |   |
     |             \    Information Field     | I |
   Span      Variable                         |   |
    of             \                          |   |
 Information       |                          |   |
    FCS            v                          |   |
   Check           --------------------------------
     |        16   ^                          |FC3|
     |        Bits |   Information Frame     -|---|
     v             v  Check Sequence Field    |FC4|
    -----          --------------------------------
```

Figure B-2 - Response Message Structure

B2.2.2 Destination Address Field (T)

The Destination Address Field (T) is an 8-bit quantity that uniquely identifies the destination of the message. No provision is made for a global destination address.

B2.2.3 Function Field (FUN)

The Function field (FUN) is used to convey the commands and responses necessary to control the data link. The most significant bit of the function field is reserved for link control as follows:

```
   0  1                 7
 ---------------------
 |  |                 |
 ---------------------
  |
  |----> = 1  For Response frame
  |
  -----> = 0  For Command frame
```

The remaining bits of the function field are available for specified commands and responses as shown below.

B2.2.3.1 Command Function

The following command functions are used in controlling the serial channel.

- Master Clear - master clears hardware.
- Downline Load - loads information field to memory without processor control.
- Downline Dump - constructs a response field from memory without processor control.
- Go - issues PDC processor go command.
- Status - requests hardware status from the serial channel interface.
- Resync - resets resync counters.

The following command functions are used to transmit information between higher level processes.

- Data - loads information field to memory with processor control.

B2.2.3.2 Response Function

The response functions are:

- Hardware Response - response generated by the serial channel hardware. It contains no I field and requires no processors interaction.
- Processor Response - response generated by the PDC processor. It contains an I field.

B2.2.4 Access Code Field (A1,A2)

The Access Code Field (A1,A2) contains a 16-bit quantity that defines the set of units which will accept this command message. The access code is a "key" which must match the "lock" on the receiving unit in order that the message be accepted. If the match is not made, the frame is discarded.

NOTE: The access code field is not defined on response messages.

B2.2.5 Resync Parameter Field (RP)

The Resync Parameter Field (RP) contains the 8-bit value to which the rotating priority counters are to be reset.

NOTE: The resync parameter field is not defined in response frames.

B2.2.6 Source Address Field (S)

The Source Address Field (S) is an 8-bit quantity that identifies the unit which sent the message.

B2.2.7 Length Field (L1,L2)

The Length Field (L1,L2) is a 16-bit quantity which defines the number of 16-bit words in the optional information field which follows the header portion of the message. If the length is zero no information field exists in this message.

B2.2.8 Header Frame Check Sequence Field (FC1,FC2)

Each message includes a 16-bit header frame check sequence (FCS) immediately following the L1,L2 field. FC1,FC2 always occurs as the ninth and tenth bytes following the frame synchronization sequence. The FCS field serves to detect errors induced by the transmission link and to validate transmission accuracy. The 16-bit FCS results from a mathematical computation on the digital value of all binary bits in the frame following the frame synchronization sequence.

The process is known as cyclic redundancy checking using the CCITT Recommendation V.41 generator polynomial of $X^{16} + X^{12} + X^{5} + 1$. The transmitter's 16-bit remainder value is initialized to all ones before a frame is transmitted. The binary value of the transmission is premultiplied by X^{16} and then divided by the generator polynomial. Integer quotient values are ignored and the transmitter sends the complement of the resulting remainder value, high order bit first, as the FCS field.

At the receiver the initial remainder is preset to all ones and the same process is applied to the serial incoming bits. In the absence of transmission errors the final remainder is 0001110100001111 (X^{15} through X^{0} respectively).

The receiver will discard a message in error. Subsequent retransmission of the discarded message is under control of error recovery procedures, to be defined later.

B2.2.9 Information Field (I)

The Information Field (I) contains data which are transmitted between higher level processes that exist in units attached to the serial channel.

The data link control is completely transparent to the contents of the I field. The I field could therefore, consist of any number of bytes, in any code, related to any character structure and limited only by system requirements. The length of the I field could be unrestricted, however, it should be recognized that typical length is contingent on system requirements and limitations beyond the link level. Factors limiting I field length include channel error characteristics, adapter buffer size, and the logical properties of the data.

The length of the Information Field is restricted to be an even number of 8-bit bytes. A length of zero is specifically permitted.

B2.2.10 Information Frame Check Sequence Field (FC3,FC4)

If the Length Field (L1,L2) is non zero, the 16-bit Information Frame Check Sequence Field (FC3,FC4) is present. The information frame check sequence is computed on all of the binary bits in the Information Field (I). The same generation polynomial that's used for FC1,FC2 is also used for FC3,FC4.

B2.2.11 Parameter Fields (P1,P2,P3)

The Parameter Fields are only defined for response messages. They are used to convey status about the received command message and the current state of the PDC TCU/TCI/Processor/DI hardware.

B2.3 Additional Conventions

B2.3.1 Intermessage Time Fill

Intermessage time fill may be transmitted to maintain the serial link in an active state to avoid timeouts and/or to hold the authority to transmit. When used, intermessage time fill must be a series of contiguous ones followed by a frame synchronization character (F) preceding the message to be transmitted.

B2.3.2 Abort

Abort is the process by which the sender, in the act of transmitting a message, decides before the end of that message to terminate in an unusual manner which will cause the receiver to discard the message. There is no special Abort character in LCNCP. In order to abort a frame the sender must terminate the transmission before transmitting the proper FCS character.

B2.3.3 Invalid Message

An invalid message is defined as a message that is too short, e.g., less than the minimum length. The minimum length is 10 characters after the frame synchronization sequence.

B2.3.4 Order of Bit Transmission

The order of transmission for all fields is most significant bit first.

B2.3.5 Compatibility

The LCP protocol is not intended to be compatible with any other protocol including SDLC, ADCCP and CDCP. The LCNCP is intended for use in systems structures, specifically excluded from SDLC, ADCCP and CDCP.

B3.0 DESIGN GUIDELINES

The following are the design guidelines used in initial implementation of the LCNCP protocol:

- Only one frame per message.
- No bit stuffing - byte synchronization is obtained by using a frame synchronization sequence rather than a unique character, a la SDLC.
- No special characters such as Abort or Flag to generate/detect.
- Trunk level and higher level protocol are truly divided - the header portion of a message contains only the trunk control data. The information field of a message contains only higher level protocol data.
- Both the header and information fields contain an even number of bytes.
- All transactions on the trunk consist of a command message and response message pair.

- Streaming is allowed - the trunk is held before command or response messages by transmitting a long-frame synchronization sequence.
- Messages can be aborted - messages are aborted by terminating the transmission before sending the FCS characters. An abort sequence can be initiated by a hardware malfunction such as parity error, a hardware master clear or a software master clear.

APPENDIX C
PROPOSED LCN UNIFIED SECOND LEVEL PROTOCOL

INTRODUCTION

A PDC contains both device and network controlware. Device controlware interfaces to the attached equipment, e.g., host machines, disk controllers, communication lines, etc. Network controlware controls the trunk hardware, and thus interfaces with all PDCs on the network which share trunks. Network controlware acts as the message delivery agent for device controlware. The device/network controlware sets interface each other by means of buffers, flags, and procedures.

The method by which network controlware acts as a message courier for device controlware is herein called the second level protocol.

DESCRIPTION

The first level LCN protocol (FLP) defines the format of trunk messages as interpreted by the TCU/TCI hardware (figure C-1). First level protocol includes routing information, a data field, and checksums. The routing information specifies which PDC sent the message and which PDC will accept it; the checksums verify the legality of the message. The existence of the data field is known by the hardware, but not the meaning of its content.

```
-----------------------------------------
|        Header         | C |  Data  | C |
| (Resync and Routing)  | S |  Field | S |
-----------------------------------------
```

Figure C-1. First Level Protocol Message

The proposed second level protocol message is contained within the FLP data field (figure C-2). It consists of a header and a data field. The header includes function, sequence number, and other pertinent information; the data field is used to convey information for higher level protocols.

```
|<--  First level protocol data field -->|
------------------------------------------
|        Header                 |        |
| (Function, Seq #, etc.)       | Data Field |
------------------------------------------
```

Figure C-2. Second Level Protocol Message

The second level protocol consists in part of a sequence of messages limited by convention to a predefined order by function and sequence number. Mutual control and synchronization between cooperating PDCs is provided by these fields.

Message sequences are usually transmitted in streaming mode.

In figure C-3 the boxes pointing to the right represent command messages sent by a source to a destination PDC. The boxes pointing left represent the response messages returned by the destination.

- REQUEST - Solicitation of destination PDC resources
- GA - Go Ahead (resources granted)
- DATA - Presenting data to the destination
- DATA/EOP - Final data message
- ACK - Acknowledgement

```
 ______________   _____   _____   _____   _____      __________   _____
| REQUEST /    \ GA  |  |DATA/ \ GA  |  |DATA/ ...  |DATA/EOP/ \ACK|
```

Figure C-3. Unified Second Level Protocol Message Sequence

The second level protocol also specifies a range of buffer sizes, reflecting requirements which vary from short requests (read disk) through medium size transfers (PRUs, symmetric link) up to high bandwidth activities (streaming 819s). In addition, flags and procedures are defined by which the host and network controlware inform each other of the state of the buffers.

Second Level Protocol Modes

Mode 1

Message sequence

```
 ____________   _____
| MODE1 REQ /  \ ACK |
```

- not stream mode
- block size: 512 bits
- no device interaction

Mode 1 consists of one command/response message pair. The message is accepted if the Mode 1 queue is not full. Typical use: high level control or function (read disk, time of day, etc.).

Command Messages

Mode 1 request - the command message is itself to be placed in the Mode 1 queue.

Response Messages

ACK - accepted and placed into the queue

NAK - queue full

BUSY/ERROR - various

<u>Mode 2</u>

Message sequence

```
SU                     SD
 |                      |
 v                      v
 ____________   ______   ____________   ______
|            \ /      | |            \ /      |
| MODE2 REQ  / \  GA  | | DATA/EOP   / \ ACK  |
 ‾‾‾‾‾‾‾‾‾‾‾‾   ‾‾‾‾‾‾   ‾‾‾‾‾‾‾‾‾‾‾‾   ‾‾‾‾‾‾
```

SU = source enters stream mode

SD = source exits stream mode

- source (only) in stream mode
- block size = 2K bytes (roughly 4 PRUs)
- no device interaction

Mode 2 consists of two command/response message pairs. The request asks for room in the Mode 2 queue; the Go Ahead means room is available. The DATA/EOP command message is the data; the ACK means the data has been accepted. Typical use: symmetric link, PRU transfer, etc.

Command messages

Mode 2 request- request for mode 2 queue

DATA/EOP - last data block (only one data block allowed)

Response messages

NAK - mode 2 queue full

GA - ready for data

BUSY/ERROR - various

ACK - acknowledgement

Mode 3

Message Sequence

```
                  SU                      SD
                  |                       |
                  v                       v
                  ______________   ______  _____________   ______
                  |             \ /      | |             \ /      |
                  | MODE3 REQ   / \  GA  | | DATA/EOP    / \  ACK |
                  ______________   ______  _____________   ______
```

- block size 4K (roughly 8 PRUs, one 819 sector)

Mode 3 is identical to Mode 2 except for the request function, the block size, and the queue (Mode 3 queue). Typical use: 819 single-sector transfers.

Mode 4

Message sequence

```
SU                                                       SD
|                                                        |
v                                                        v
___________   _____   _____   _____   _____        ________   _____
|          \ /     | |     \ /     | |     \...|           \ /     |
|MODE4 REQ/ \ GA   | |DATA/ \ GA   | |DATA/   |DATA/EOP/ \ACK|
___________   _____   _____   _____   _____        ________   _____
               ^                                              ^
               |                                              |
               DU                                             DD
```

SU = source enters stream mode

SD = source exits stream mode

DU = destination enters stream mode

DD = destination exits stream mode

- both source and destination in stream mode
- block size = 4K bytes
- device interaction

Mode 4 may have virtually an unlimited set of command/response message pairs. Typical use: streaming an 819; 176/176 file transfers.

Command messages

MODE4 REQ - request to enter mode 4

DATA - nth block in m block transfer

DATA/EOP - last block

(ABORT) - source aborting

Command responses

GA - PDC <u>and</u> device ready for mode 4, and room for next data block

(WAIT) - device agrees to mode 4, but all resources not yet ready

ACK - mode 4 normal close out

(ABORT) - destination aborting

ERROR - various

NAK - PDC disallows mode 4

<u>General</u>

- Modes 1, 2, and 3 all request their respective queue resources at the destination. None of these modes require device interaction.
- A given PDC may or may not have all four modes in its function set (i.e., mode 4 not required on lower CYBER). However, all would have at least Mode 1.
- Response set including exception conditions by mode (network controlware generated).