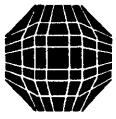
CRAY-3 Hardware Reference Manual

CSOS 1.0 Publication Number: 3202



Cray Computer Corporation
1110 Bayfield Drive
Colorado Springs, CO 80906

Copyright

Revision History

Release Date	Summary of Changes
Nov 8, 1991	Initial Release of Manual.
Feb. 1, 1994	Updated to reflect background processor issue conditions and timings in phases. Replaced memory descriptions to reflect 4 Megabit SRAM chips. New LSX interface description. Added HiPPI desciption. Minor corrections and additions throughout the manual. Made minor changes to formatting.

Reader Comments

If you have any comment about the technical accuracy, content, or organization of this manual, please tell us. You can contact us in any of the following ways:

Call our Technical Publications department at (719) 579-6464 during the hours of 8 a.m. to 5 p.m. (Mountain Time).

Send us electronic mail from a CSOS or UNIX system to pubs@craycos.com.

Write to us at our address:

```
Cray Computer Corp.
Technical Publications Department
1110 Bayfield Drive
Colorado Springs, CO 80906
```

We value your comments and will respond to them promptly.

Table of Contents

List of Figures

Conventions

An effort has been made to utilize the following conventions throughout Cray Computer Corporation (CCC) manuals and documentation:

Typographic:

Body text is set in 12-point Times Roman font.

boldface	**Boldface** indicates any literal value or name, including CSOS commands, command options, file names, and directory names which a user is expected to type verbatim.
italics	*Italics* represent terms being defined, words or names of variables for which the user supplies exact information and for emphasis.
`typewriter`	Attributes, procedures, macros, and anything resembling source language code are all set in a `typewriter` font. Representations of anything that might appear on your screen are also set in a `typewriter` font.
UPPERCASE	Uppercase is primarily used for programming language statements or functions (e.g., Fortran statements) and acronyms.
underscore	Underscored words in command lines indicate default values; underscoring is also used to specify accepted option abbreviations.
■	■ is the symbol indicating a Note.
▼	▼ is the symbol indicating a reference to another manual.
◆	◆ is the symbol indicating a Caution or Warning.
[ ]	Brackets enclose optional portions of a command or directive format.

a | b	A vertical bar in a command format separates two or more possible choices, one of which you may specify.
choice1 choice2	Stacked items indicate two or more literal parameters when only one of those parameters may be used.
name(*number*)	Items in text with a *number* in parentheses after them are references to CSOS manual page descriptions. The (*number*) denotes the section of CSOS documentation in which they are described, for example, **f77**(1) refers to the CRAY-3 Fortran user command manual page and **ctime**(3C) refers to the C library ctime function.
"*string*"	Quotation marks are used to delimit literal strings, groups of words which are used as a single word or in describing a unique concept or in the invention of new terminology.
...	Ellipses indicate the optional use of the preceding item two or more times in succession.
O'	The capital letter O, followed by an apostrophe is used to indicate an octal number; O'177777 means 177777 octal.

Acronyms:

CCC is an acronym of Cray Computer Corporation.

CRI is an acronym of Cray Research, Incorporated.

NFS is an acronym of Network File System.

SVR4 is an acronym of the System V Release 4 system of UNIX Systems Laboratories.

TCP/IP is an acronym of Transmission Control Protocol/Internet Protocol.

Nomenclature:

dataset/file	The terms *dataset* and *file* are used interchangeably in some manuals, except where a difference is explicitly noted. Under CSOS, a file specification can be a file path name.
on | off	The terms on and off are used interchangeably with enabled and disabled, true and false, or with 1 and 0, respectively.

Manual Page Formats:

Many Cray Computer Corporation manuals are comprised of information which is available on-line for use with the **man**(1) command. To retrieve a manual page entry, the following command may be typed, substituting the entry of interest for *entry*:

man *entry*

If there is more than one entry of the same name, all entries are printed. To retrieve the entry for a particular section, an optional section specification may be supplied between the man command and the desired entry:

man *section entry*

For example, the "**man write**" command will display three different man pages: one for the **write** command (section 1), one for the **write** system call (section 2), and one for the **write** Fortran I/O library function (section 3u). If the user is interested solely in the write system call, then the command "**man 2 write**" may be used. For more information on the man command and the various sections available, see **man**(1) by issuing the command "**man 1 man**".

Standard typesetting conventions for printed manual pages include:

bold	**Boldface** indicates literal strings, including command names, directory names, file names, path names, man page entry names, options, shell or system variable code names, system call names, C structures and C reserved words.
italic	*Italics* indicate variables, user-supplied (non-literal) options, terms or concepts being defined within the text and for additional emphasis.

It should be noted that these conventions are not necessarily adhered to by various text filters available on the CSOS system, e.g., **pg**(1), **more**(1). Some filters will render **boldface** indistinguishable from normal Roman body text. Some filters will render *italics* as underscored. Thus, these conventions are not immediately applicable to the visual perception of information displayed with on-line utilities and documentation.

Manual page entries are based upon a common format. The following list shows the order of sections within a manual page and provides a brief description of the information content of those sections. Not all sections shown below are found in each manual page entry.

NAME	Shows the name of the entry and briefly states its function.
SYNOPSIS	Presents the syntax of the entry. The following conventions are used in the SYNOPSIS section: Brackets [ ] enclosing a command line component may indicate that the component is optional. When an argument or operand is given as *name* or *file*, it always refers to a file name. Ellipses (...) indicate that the preceding command line component may be repeated. An argument beginning with a minus, plus or equal sign (-, +, or =) is usually an option.
DESCRIPTION	Discusses the entry and its purpose or function in detail.
OPTIONS	Lists and describes the entry's options, their purpose, use and potential interactions.
IMPLEMENTATION	Provides details for using the entry.
NOTES	Points out items of particular importance.
CAUTIONS	Describes actions which may alter data or produce undesirable results.
WARNINGS	Describes actions which may produce harmful effects on the system or its users.
EXAMPLES	Provides examples of usage for the entry.
FILES	Lists files that are either part of the entry or related to it.

RETURN VALUE	Describes return values for the entry and possible error return codes.
MESSAGES	Describes the informational, diagnostic and error messages that may be produced by the entry. Self-explanatory messages are generally not listed.
DIAGNOSTICS	Describes diagnostic messages produced by the entry.
BUGS	Lists known bugs or deficiencies of the entry.
SEE ALSO	Lists man page entries or manuals which contain information related to this manual page entry.

Chapter

1 Overview

A brief overview of the CRAY-3 hardware characteristics is presented. It is contrasted with the CRAY-2 characteristics where appropriate.

1.1 Introduction

The CRAY-3 computer system represents a major enhancement over the CRAY-2 computer system, providing an order of magnitude performance increase at a comparable cost. This performance gain is achieved through the use of gallium arsenide logic circuits in place of the silicon circuits previously used, innovative packaging and cooling technologies and a number of architectural changes, including more processors and enhanced memory access.

Each processor in the CRAY-3 system is on the order of two to three times as fast as its CRAY-2 counterpart. With a fourfold increase in the number of processors, this leads to an overall performance increase ranging from 8-12 times that of a CRAY-2.

The CRAY-3 system is functionally equivalent to the CRAY-2 system. Logic enhancements to the system have enabled performance to be increased while providing an extension of the original CRAY-2 instruction set. The following items list the major differences between the CRAY-2 and CRAY-3 systems.

1.1.1 Circuit Speed

The CRAY-3 system has a clock speed of 2 nanoseconds, twice that of the Cray-2. In addition, the use of gallium arsenide allows six levels of logic per clock as compared to four on the CRAY-2.

1.1.2 Number of Processors

The CRAY-3 system supports up to 16 Background Processors and one Foreground Processor. The CRAY-2 system offers four background processors and one foreground processor.

1.1.3 Instruction Issue Rate

The CRAY-3 processors can issue an instruction each clock period, providing for a maximum instruction issue rate of 8000 Million Instructions Per Second (MIPS) for a 16-processor configuration. The CRAY-2 processor can issue an instruction every other clock period leading to a maximum issue rate of 500 MIPS for a four-processor configuration.

1.1.4 Common Memory Size

The CRAY-3 Common Memory consists of 2048 million words of static semiconductor memory. Memory chip access time is 25 nanoseconds. The CRAY-2 common memory consists of 256 million words of dynamic semiconductor memory. Memory chip access time is 80 nanoseconds.

1.1.5 Common Memory Bandwidth

The CRAY-3 background processors each have a bi-directional common memory access port. This port allows memory reads and writes to proceed simultaneously. Total common memory bandwidth is 16 gigawords per second. The CRAY-2 background processors have one common memory access port each. This port can read or write but cannot do both at any given time. Total common memory bandwidth is one gigaword per second.

1.1.6 Floating Point Computation

The use of vector processing techniques allows the CRAY-3 background processors to compute at a maximum burst rate of 16 gigaflops. The CRAY-2 background processors can compute at a maximum burst rate of two gigaflops. The CRAY-2 rate is usually limited to around one gigaflop because of common memory bandwidth.

1.1.7 Vector Register Structure

A CRAY-3 background processor contains eight vector registers which have independent reading and writing capability. Writing into a vector register may begin immediately after the previous data read has been initiated. This feature, known as tailgating, represents a major improvement in vector register availability over the original CRAY-2 system which required read completion before write initiation.

1.2 Hardware Design

The CRAY-3 hardware is constructed of synchronous networks of binary circuits. These circuits are packaged in 336 pluggable modules. The common memory occupies 256 of these modules. Each memory module contains 144 silicon static Random Access Memory (RAM) circuit packages. Each of the 16 background processors occupies four logic modules, and the foreground processor consists of one logic module. The logic modules each contain up to 1024 GaAs circuit packages. The remaining 15 modules provide control for I/O devices such as disk drives. The total integrated circuit population in the system is approximately 140,000 of which 36,864 are memory.

The pluggable modules are three-dimensional structures measuring 4.76 inches wide by 4.22 inches high by .281 inches deep. The circuit packages within the modules are unpackaged GaAs or silicon chips mounted directly onto multi-layer circuit boards. These boards are arranged in a four-by-four matrix of board stacks, each stack four boards deep. In the center of this structure are two multi-layer logic plates which provide routing of logic signals between the 16 board stacks in the module. Between the two logic plates of the module are two multi-layer plates for power distribution to the board stacks, and the resistor plate for signal termination is in the module center between the power plates. Circuit connections are made in all three dimensions within the module.

One edge of the module consists of four power blades for connection to the system power supplies and ground. The other three edges have up to 22 twisted pair wires connected. The twisted pairs are themselves terminated in a stripline connector. The purpose of the twisted pair wire and connector is intermodule communication either directly or via a central wire mat.

Modules are arranged in the cabinet frame in eight columns which form a closed octagon. Each column contains 42 modules in an outer bank of 32 and an inner bank of 10. An electrically inert electronic circulates in the cabinet frame and flows through the modules for cooling. The temperature, liquid velocity and turbulence through the modules are controlled.

The octagon of module columns is located on top of a similar structure containing power supplies for the system. Total power consumption for the system is 450 kilowatts. Total cabinet height, including the power supplies, is 42 inches.

There are approximately 450 different GaAs integrated circuit packages used in the logical networks of the machine. Each of these circuits consists of an array of up to 128 basic logic cells. There are 20 different logic cells from which all the circuits are built. The maximum equivalent gate capacity of the circuit packages is 500. The logic circuit packages have 52 bonding pads for connection to the printed circuit boards.

A 500-megahertz oscillator controls the timing throughout the circuit modules in the machine. The oscillator signal is transmitted as a square wave over 60 ohm twisted-pair wires to each of the module connectors. Wire lengths are controlled so that the travel time to the individual modules is accurate within 50 picoseconds. The oscillator square wave is delivered to each individual circuit package requiring a clock input within the module. A pulse is formed from the square wave to gate data and control into latches within the packages. This strobe pulse occurs simultaneously throughout the machine within a period of two nanoseconds. This time is referred to as the machine clock period.

A pulse is also formed from the inverted clock signal and is used to gate information into latches one-half clock period later than the normal clock signal. This half clock period (one nanosecond) of time is referred to as a clock phase time. The normal clock signal latches information in the even clock phases and the inverted clock signal latches information in the odd clock phases. This dual phase system clock allows information to be latched into successive logic cells every nanosecond.

1.3 Common Memory

The CRAY-3 system common memory consists of 512 storage banks of four million words each. Each word consists of 64 data bits and eight error correction bits. This memory holds information shared by the foreground processor, background processors and peripheral equipment controllers. It contains program code for the background processors, as well as data for problem solution. System tables are located here for the foreground processor but foreground program code is not. Data buffers for the disk files are located in common memory.

Each memory bank occupies one-half of a CRAY-3 circuit module. A memory bank can utilize 32 background processor memory access ports, one read and one write port per processor. Total memory bandwidth is 128 gigabytes per

second. Total memory capacity is 16 gigabytes. Each background processor can read and write a word of data per clock period (two nanoseconds) in a vector mode.

The silicon Static Random Access Memory (SRAM) integrated circuits used in common memory each contain 4,194,304 bits of data. A memory bank consists of 72 of the 4M-by-1 SRAM integrated circuits. The memory module then contains 144 x 4M bits of data, plus logic circuits to support the memory access paths. Memory access time at the circuit level is 25 nanoseconds. Memory cycle time is 25 nanoseconds.

The CRAY-3 GaAs logic circuits are significantly faster than the memory circuits. This speed discrepancy is used to minimize the number of physical data paths that are necessary to connect the 512 memory bank modules to the 32 memory ports. Data is transferred by utilizing an 18-bit packet. The 64-bit data word and eight error correction bits require a four-clock period transmission time, 18 bits per clock period. There are independent read and write packets between the memory bank and access port. Memory bank address utilizes a 12-bit packet. The 24-bit internal bank address requires a two clock period transmission time between the access port and memory bank, 12 bits per clock period.

An eight-bit error correction code is generated at the memory access port as the write data is transmitted to the memory bank. This code is interpreted at a read-out port for single error correction and double error detection.

1.4 Background Processors

The 16 background processors in the CRAY-3 system are composed of processing elements similar to those in a CRAY-2 processor. The CRAY-3 instruction set includes similar registers and arithmetic functions, and all the CRAY-2 instructions have been retained. Some of the instruction source and destination register designators have been changed to provide consistency and logical simplification. Eight new instructions have been added.

Briefly, they are:

Machine Code	CAL Instruction		Description
001---	CMR		Complete memory references
023ij0	A_i	A_j	Copy A_j to A_i
023ij1	A_i	A_j+1	Increment A_j to A_i
023ij2	A_i	A_j-1	Decrement A_j to A_i
035-j-	RT	S_j	Copy S_j to RT (Monitor Mode only)
074i--	V_i	[lm]	Direct read V_i from Local Memory
075i--	[lm]	V_i	Direct write V_i to Local Memory
134ijk	V_i	$<A_j,A_k>$	Two-port read V_i from CM (A_j) stride A_k
135ijk	$<A_j,A_k>$	V_i	Two-port write V_i to CM (A_j) stride A_k
177---	PASS		No-op

Computation in a background mode implies a memory-to-memory computation. A job is initiated by the foreground processor. Data is moved from the disk files to the common memory under foreground processor supervision. The program code for the background processors is positioned in the common memory, and the computational field length is defined. The foreground processor then initiates the background computation using one or more background processors as required. The background processors may call for further peripheral activity through the foreground processor as the computation proceeds.

Each background processor has a small, high-speed local memory to hold scalar or vector operands during a computation. Data is moved from the common memory to the local memory and returned at the end of each computation. Arrays of data are addressed by the background processors directly in the common memory. The access to data common to multiple background processors is interlocked by the background processor semaphore flags.

Resources of a background processor are summarized below:

Eight S registers	64 bit length	For scalar operands
Eight A registers	32 bit length	For address and integer operands
Eight V registers	64 elements of 64 bits	For vector segments
Instruction stack	512 parcels of 16 bit length	For instruction loops
Local memory	16384 words of 64 bit length	For A, S and V register backup

Eight S registers	64 bit length	For scalar operands
Floating point functional units		For computation in 64 bit floating point mode
Integer functional units		For computation in 64 bit integer mode
Address functional units		For computation in 32 bit integer mode
Vector and Scalar logical units		For 64 bit logical computations
Vector and Scalar shift units		For shifts, population and leading zero counts

1.5 Foreground Processor

The foreground processor supervises overall system activity and responds to requests for interaction between the system members. System communication is accomplished through four high-speed synchronous data channels. These channels interconnect the background processors, foreground processor, disk control units and host system interfaces.

Instructions for execution in the foreground processor are loaded into a special memory at system deadstart time from a file on the maintenance console. This memory then becomes a read-only memory during system operation. Data for supervision of the system is maintained in the common memory and is moved to the foreground processor local data memory as required.

The majority of foreground processor activity involves data transfer between the disk file storage units and the common memory. The system provides for a mixture of 12 megabyte per second disk file storage unit interfaces and 100 megabyte per second interfaces to mass storage subsystems such as disk arrays.

Resources of the foreground processor are summarized below:

Local data memory	4096 words of 32 bit length	Temporary storage
Instruction memory		65,536 bytes of 8 bit length arranged in 16 banks
Integer functional units		For computation in 32 bit integer mode
Four communication channels		One gigabyte per second each

1.6 Disk Systems

1.6.1 RD Disk Array Subsystem

Cray Computer utilizes the latest in rotating mass storage technology, Redundant Array of Inexpensive Disks (RAID) units. The RAID technology meets the performance requirements of high-performance computer systems as well as the requirements for data preservation, data availability, storage capacity and cost/performance.

Data preservation and data availability are accomplished on the RAID technology by storing data across an array of multiple disk drives. For each block of data a "parity block" is computed and stored on an independent disk drive. In the event of a disk drive failure, the RAID can continue to read and write data utilizing the remaining disks. With a mean time to interrupt of 150,000 hours per disk drive the RAID technology provides a calculated mean time to data loss for the RD subsystem of 34 million hours (almost 4,000 years).

RAID-1 technology uses mirrored disk drives to ensure data preservation and data availability. RAID-3 technology stripes data by bytes across its array of disk drives, using all the disk drives in the array, including the "parity drive." RAID-5 stripes data by data B blocks across the array of disk drives using only the number of drives required for the data, plus the "parity block." The RD disk units provide higher performance disk drives than are normally available on supercomputers. Therefore, the RD units are more efficient for large data block transfers, application program files, swap files, etc. In addition, the RAID-5 technology provides efficient smaller data block transfers normally associated with operating system files.

Additional data preservation and data availability is accomplished with a "hot-spare" drive. In the event that a disk drive fails it is taken off-line by the operating system. The hot-spare drive replaces the failed drive, and the controller reconstructs the data located on the failed drive to the hot-spare using the data on the remaining disks. The computer system has read/write access to RAID during the reconstruction period. With the completion of the data reconstruction the RAID unit can sustain another disk drive failure without loss of data or read/write access. The failed disk drive can be removed and replaced while the RD subsystem continues to operate, thus restoring the hot-spare capability.

Data preservation and data availability, along with performance, are extremely important in UNIX environments where loss of a single data block can cause an entire file system to fail.

All the RAID disk units are directly connected to the CRAY-3 by a dedicated 32-bit HIPPI channel. Therefore, each RAID unit operates independently from any other RAID, allowing overlap of reading, writing and seeking. However, indirect connection of RD subsystem to the CRAY-3 through a HIPPI switch is also supported.

The RD unit controller provides an internal data buffer and command queueing that greatly enhances the performance of the unit. The data buffer allows data transfers between the CRAY-3 and the buffer to operate asynchronously with the data transfer between the buffer and the disk drives. In addition, the controller supports simultaneous reads and writes of data from the internal data buffer tot he CRAY-3 via the HIPPI channel. For write commands command queueing allows data to be transferred from the CRAY-3 to the internal buffer before the previous commands are completed. For read commands command queueing also allows additional commands to be processed while the data is being transferred from the internal buffer to the CRAY-3.

In RAID-5 technology command queueing is extended to another level of overlapped operation. RAID-5 technology allows each disk drive in the array to begin operation on the next command as soon as the disk drive has completed the previous command without waiting for the entire previous command to complete.

The CRAY-3 operating system also supports command reorder, which attempts to optimize the disk by rearranging the order of commands to minimize disk seek time.

The available models of the RD subsystem are:

Model	No. of HIPPI Channels	Capacity Gigabytes
RD-41/24	1	11
RD-42/24	2	11
RD-41/48	1	23
RD-42/48	2	23

- An RD-41 may be upgraded to an RD-42.
- An RD-41/24 may be upgraded to an RD-41/48 and an RD-42/24 to an RD-42/48.

The performance of the RD/41 and RD-42 disk units is.

Transfer Type	Megabytes/Second
Maximum transfer rate (read)	86.0

Chapter

2 Common Memory

A detailed description of the common memory and its interface to the background processors is presented.

2.1 Organization

The common memory shared by the background processors uses a four-megabit Static Random Access Memory (SRAM) storage device and provides from 512 to 2048 million words of directly accessible memory depending, upon the system configuration. Individual banks provide 4,194,304 words of storage. Each word of storage is 72 bits long, consisting of 64 bits of data and an eight-bit single error correction, double error detection code. The 512 independent banks of common memory are divided into memory octants corresponding to the eight physical octants in the computer system. There are 64 banks of memory in each memory octant. Each bank of memory occupies half of a memory module.

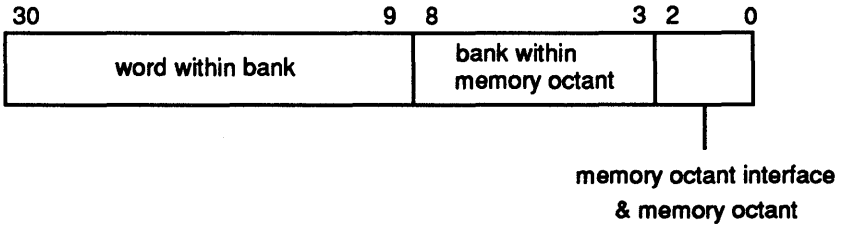
Access to common memory by the background processors is controlled by a combination of common memory octant interfaces, retry buffers within the background processors and buffer segments within each memory octant. Each background processor contains eight common memory octant interfaces, one for each memory octant. There is also a retry buffer capable of containing 64 write and 64 read references in each background processor. Each memory octant contains four buffer segments for retaining references from the 16 background processors. Each buffer segment handles references for four of the

background processors.The background processors make references to the memory octants in a time structure which allows a processor access to a memory octant once every four clock cycles. The access mechanism which controls this structure is referred to as the common memory slot time. A memory bank requires 13 clock periods to service each memory reference once the reference is forwarded to the bank by the buffer segment.

The connection between a background processor and a memory octant consists of various control lines, a 19-bit wide address path, an 18 bit-wide write data path and an 18-bit wide read data path. Transmission of a 72-bit word between a background processor and a memory octant requires four clock periods, yielding a bandwidth of 125 megawords per second. There are 128 of these connections between the eight memory octants and 16 background processors. They provide a maximum common memory bandwidth of 16 gigawords per second. Each background processor is capable of submitting two memory references to different memory octants in a single clock period. The maximum possible demand for memory bandwidth is one gigaword per background processor or 16 gigawords for the 16 background processors in the computer system.

2.1.1 Addressing

The bits of an absolute word address are used by a background processor to determine the memory octant interface, bank selection and word within a bank for a reference. The usage of the bits in the address is as follows:



For various system configurations the hardware bank and octant designations are generated by rearranging the low-order bits of an address. The address bits are designated from highest (2^{30}) to lowest (2^{0}).

Cpus	Octants	Unused	Chip Address	Bank Pointer	Octant Select
2	1	30 29 28	27 26 25 ... 8 7 6	0 1 2 5 4 3	
4	2	30 29	28 27 26 ... 9 8 7	0 1 6 5 4 3	2
4	4	30	29 28 27 ... 10 9 8	0 7 6 5 4 3	2 1

Cpus	Octants	Unused	Chip Address	Bank Pointer	Octant Select
8	4	30	29 28 27 ... 10 9 8	0 7 6 5 4 3	2 1
16	8		30 29 28 ... 11 10 9	8 7 6 5 4 3	2 1 0

2.1.2 Address Generation

All common memory references occur as single-word read or write references. Single-word references are generated by independent read and write address generators to service:

- scalar common memory references
- vector single-port common memory references
- vector dual-port common memory references
- instruction field fetches
- foreground buffer operations

Operand references and instruction field fetches are added to the processor's Base Address (BA) and tested against the background processor's limit address during address generation. Invalid references cause the background processor to interrupt with a common memory range error, if interrupt on common memory range error is selected in the background processor status register.

The address generators operate as functional units and hold issue while reserved for an earlier operation. Other conditions can also delay some operations:

- Scalar references and single-port vector references wait for earlier single- and dual-port vector references to clear the address generators.
- Dual-port vector references wait for earlier single-port vector references to clear the address generators.
- Operand references take priority over instruction field fetch, which takes priority over foreground buffer operation.
- The Complete Memory References (001) and Clear Semaphore (007) instructions hold issue until address generation, the retry buffer and all eight of the background processor memory octant interfaces are empty. Common memory banks activated by recent references may still be busy.

2.2 Background Processor Common Memory Interfaces

The eight memory octant interfaces of a background processor operate independently of each other. References from a single background processor arrive at different memory banks independently, and data read from memory may arrive at the background processor out of the order that the references were submitted to the memory octants. Logic circuits within the background processor interfaces track the references and emit the proper destination tags as read data returns from the memory octants.

The background processor memory octant interface to receive a reference is determined by the low-order bits of the memory address. Each background processor memory octant interface consists of a three-entry buffer referred to as ranks A, B and C. When a reference is submitted to a memory octant interface, it will enter the furthest entry in the buffer that is currently free (C, B, or A). See Figure 1 on page 16.

When rank A of a memory octant interface is busy and an additional reference is received, the reference is deferred to one of two retry buffers (one for read references and one for write references), and a hold issue condition occurs in the background processor. The retry buffers are shared by the eight common memory octant interfaces within a background processor and can contain 64 write and 64 read common memory references. Address generation for memory references that are already active because of vector memory references, instruction field fetches or foreground buffer operations are allowed to continue until complete. The background processor then waits for all eight of its memory octant interfaces to empty. The entries in both retry buffers are then retried at a rate of one per clock period. In case of simultaneous read and write references to the same memory octant interface, read references have priority over write references. When all entries in the buffers have been retried, the buffers are checked to see if any of the references were returned to them because of backup conditions in the background processor memory interfaces. The retry process is repeated until the buffers are empty. The background processor continues to hold issue until all eight of its memory octant interfaces are empty (all references have been accepted by memory octants).

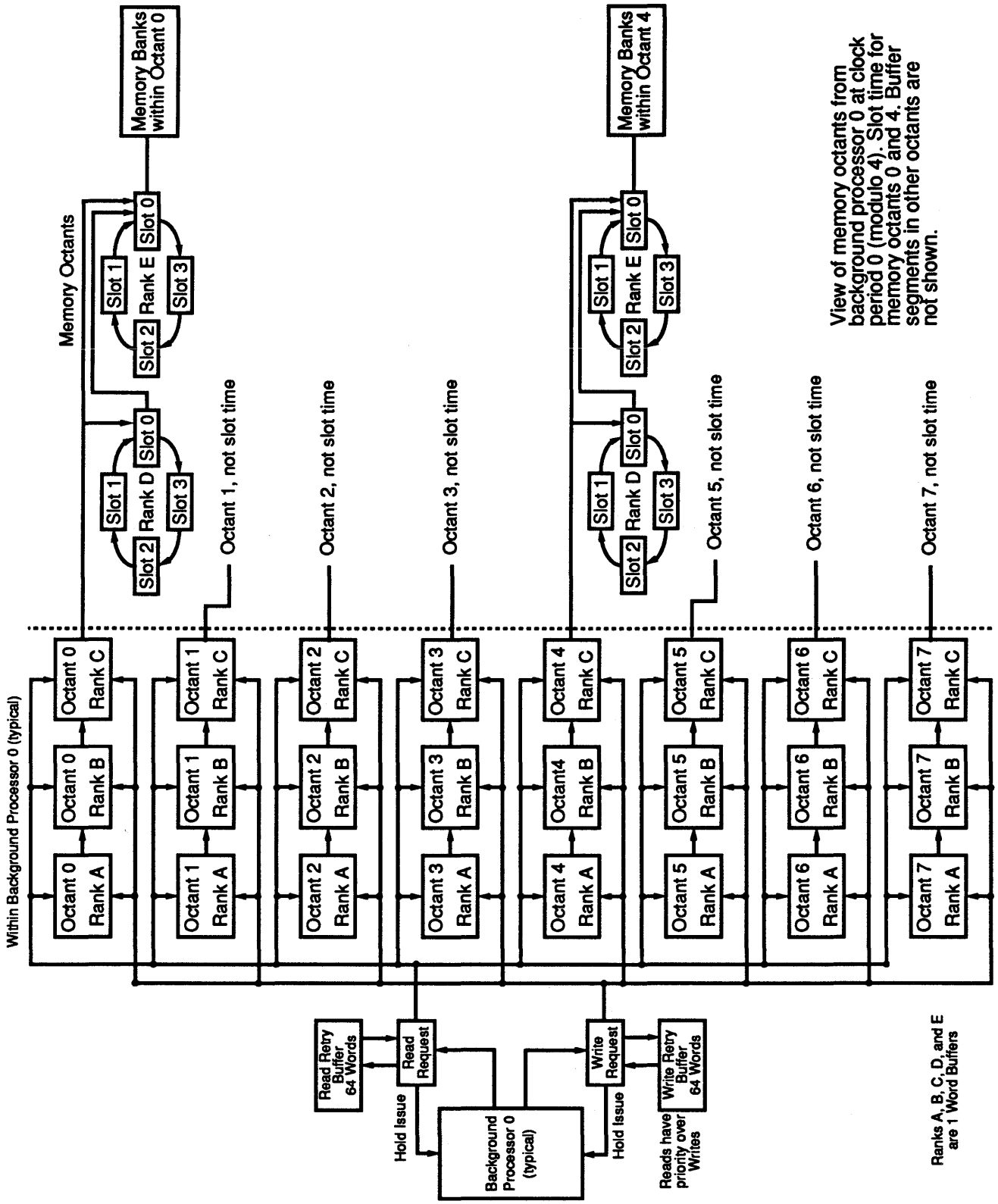
A background processor is allowed to submit references to two memory octants during each clock period. The memory octants (O) allowed to receive references from a background processor in a clock period are determined as a function of the processor number (P) and the clock period (C). Arithmetic in calculating the memory octants is performed modulo 4 and identifies a pair of octants 4 apart. The lowest octant number in the pair (modulo 4) is given by:

$$O = C + P$$

The following table indicates which memory octants may accept references from each background processor in each clock period:

Processor	Clock Period (modulo 4)			
	0	1	2	3
0, 4, 8, 12:	0, 4	1, 5	2, 6	3, 7
1, 5, 9, 13:	1, 5	2, 6	3, 7	0, 4
2, 6, 10, 14:	2, 6	3, 7	0, 4	1, 5
3, 7, 11, 15:	3, 7	0, 4	1, 5	2, 6

Figure 1 Background Processor Memory Octant Interfaces



2.3 Memory Octant Operation

Each memory octant contains four buffer segments for retaining references from the 16 background processors. Each buffer segment controls access from four background processors. See Figure 2 on page 18. Each buffer segment can receive a reference from one of the four background processors it interfaces in a clock period. Only one of the background processors attached to a buffer segment is allowed to submit a reference to the buffer segment in a clock period. The background processors (P) allowed to submit references to a memory octant in a clock period are determined as a function of the octant number (O) and clock period (C). Arithmetic in calculating the background processors is performed modulo 4 and identifies a set of four background processors 4 apart. The lowest background processor number in the group (modulo 4) is given by:

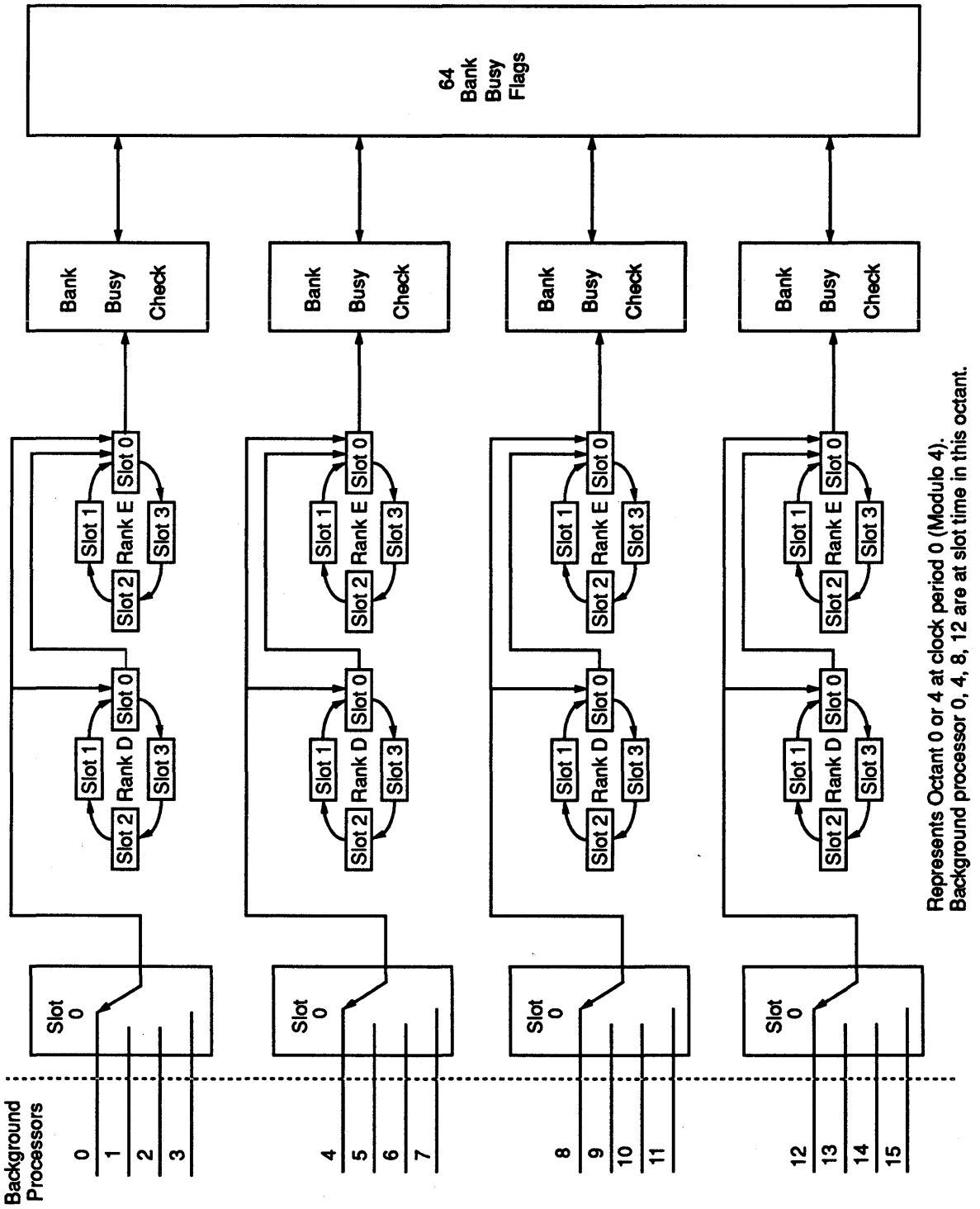
$$P = O - C$$

The following table indicates which background processors may submit references to each memory octant in each clock period:

Octant	Clock Period (modulo 4)			
	0	1	2	3
0, 4:	0, 4, 8, 12	1, 5, 9, 13	2, 6, 10, 14	3, 7, 11, 15
1, 5:	1, 5, 9, 13	2, 6, 10, 14	3, 7, 11, 15	0, 4, 8, 12
2, 6:	2, 6, 10, 14	3, 7, 11, 15	0, 4, 8, 12	1, 5, 9, 13
3, 7:	3, 7, 11, 15	0, 4, 8, 12	1, 5, 9, 13	2, 6, 10, 14

A segment buffer contains primary and secondary buffers. The primary buffer (referred to as Rank E) can have four references revolving in a four-clock-period rotating sequence, each reference entered by a background processor during its given clock period. The secondary buffer (referred to as Rank D) is utilized when a memory octant receives a reference from a background processor and the primary buffer contains a pending reference for the same background processor. The secondary buffer also can have four references revolving in a four-clock-period rotating sequence, each reference entered by a background processor during its given clock period.

Figure 2 Buffer Segments within a Memory Octant



When the memory octant receives a reference from a background processor, the reference is inserted into the primary buffer. The bank being referenced is then decoded and a check for bank busy is performed. If the bank is busy the reference will remain in the primary buffer and be retried four clock periods later. If another background processor controlled by a different buffer segment has also entered a reference for the same bank during this clock period, the lowest numbered background processor will receive priority. All remaining references will remain in the primary buffers and be retried four clock periods later.

Any additional reference received by a buffer segment that contains a pending reference in the primary buffer from the same background processor will enter the secondary buffer for that processor.

When the pending request in the primary buffer advances to the memory bank, any reference in the secondary buffer will advance to the primary buffer. An acknowledge is sent to the background processor memory octant interface, allowing any references in its three entry buffer to advance one rank (C to memory octant, B to C, A to B), and the bank busy flag is set for 13 clock periods.

2.4 Performance Considerations

Note that a vector common memory reference with a stride of 1 will submit one reference to each background processor memory octant interface every eight clock periods. A vector common memory reference with a stride of 2 (such as generated for Fortran COMPLEX or DOUBLE PRECISION data) will submit one reference to alternate background processor memory octant interfaces every four clock periods. After an initial start-up period references will be transferred from the background processor memory octant interfaces to the memory octant reference buffers at a rate of one per clock period, and (barring bank conflicts with other memory references) stride 2 vector memory references will perform at the same rate as stride 1 memory references. Vector references with strides of 4, 8, etc. should be avoided if possible to lessen the chance of background processor memory octant interface backup.

2.5 Error Correction and Detection

The eight-bit Single-Error Correction, Double-Error Detection (SECDED) code that is part of each 72-bit common memory word can reliably correct single-bit errors and detect double-bit errors in the 64-bit data portion of the word. Errors in three or more bits yield ambiguous results. Independent logic

circuits in each background processor create SECDED codes to accompany write references and interpret the codes returned with read data.

Each bit in the SECDED code is generated by computing the exclusive OR of the data bits which affect that SECDED bit. The following table shows which SECDED bits are affected by each data bit. Data bit 63 is the high-order bit of the data, and data bit 0 is the lowest-order (least significant) data bit.

	2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}
Check Bit 0									x	x	x	x	x	x	x	x
Check Bit 1	x	x	x	x	x	x	x	x								
Check Bit 2	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Check Bit 3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Check Bit 4	x		x		x		x		x		x		x		x	
Check Bit 5	x	x			x	x			x	x			x	x		
Check Bit 6	x	x	x	x					x	x	x	x				
Check Bit 7	x			x		x	x		x			x		x	x	

	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}
Check Bit 0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Check Bit 1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Check Bit 2									x	x	x	x	x	x	x	x
Check Bit 3	x	x	x	x	x	x	x	x								
Check Bit 4	x		x		x		x		x		x		x		x	
Check Bit 5	x	x			x	x			x	x			x	x		
Check Bit 6	x	x	x	x					x	x	x	x				
Check Bit 7	x			x		x	x		x			x		x	x	

	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
Check Bit 0	x		x		x		x		x		x		x		x	
Check Bit 1	x	x			x	x			x	x			x	x		
Check Bit 2	x	x	x	x					x	x	x	x				
Check Bit 3	x			x		x	x		x			x		x	x	
Check Bit 4									x	x	x	x	x	x	x	x
Check Bit 5	x	x	x	x	x	x	x	x								
Check Bit 6	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Check Bit 7	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{09}	2^{08}	2^{07}	2^{06}	2^{05}	2^{04}	2^{03}	2^{02}	2^{01}	2^{00}
Check Bit 0	x		x		x		x		x		x		x		x	
Check Bit 1	x	x			x	x			x	x			x	x		
Check Bit 2	x	x	x	x					x	x	x	x				
Check Bit 3	x			x		x	x		x			x		x	x	
Check Bit 4	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Check Bit 5	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Check Bit 6									x	x	x	x	x	x	x	x
Check Bit 7	x	x	x	x	x	x	x	x								

Each SECDED bit is the exclusive OR of 40 data bits, and each data bit affects an odd number of SECDED bits.

When data returns from a data reference, a check code is computed from the data and compared with the SECDED code accompanying the data. The exclusive OR of the check code and the SECDED code yields an eight-bit syndrome code. The syndrome code indicates one of the following conditions:

- If the syndrome code is clear, no error is assumed.
- If a single syndrome code bit is set, that SECDED bit is in error.
- If an odd number of syndrome code bits are set, a single-bit error has occurred and is corrected by the background processor common memory interface logic.
- If an even number of syndrome code bits are set, a double-bit error is assumed.
- A failure of three or more data bits will be misinterpreted as a single-bit error if an odd number of syndrome code bits are set.

Example of a single-bit error: A failure in bit 32 will set syndrome code bits 2, 1, and 0. A single-bit error will be assumed and data bit 32 complemented to correct the error.

Each background processor has an error data register. If the appropriate bits are set in the error status register, an interrupt will occur when either a single- or double-bit error occurs. The syndrome code is entered into the background processor error data register along with indicators for single- and double-bit errors.

Chapter

3 Background Processor

A detailed description of the background processor and its instruction set is presented.

3.1 General Description

The background processors are intended for floating-point computation in a 64 bit mode. Significant speed improvements occur when the computation can be organized into vector streams. The structure of the background processors is designed to enhance the vector capability of the system. Resources of a background processor can be summarized as follows.

3.1.1 Vector (V) Registers

There are eight vector registers in a background processor. Each register contains 64 words of 64 bit length. These registers are used as a computational way station for data between the memory and the functional units.

3.1.2 Scalar (S) Registers

There are eight scalar registers in a background processor. Each register contains a single 64 bit word. These registers are used to support the vector registers in vector streaming where one element of the computation is a constant value. The scalar registers are used as computational way stations

between the memory and the functional units where vector implementation of the work is not possible.

3.1.3 Address (A) Registers

There are eight address registers in a background processor. Each register contains a single 32 bit word. These registers are used to calculate memory locations for both the local memory and the common memory. They are also used to compute integer results in a 32 bit mode.

3.1.4 Local Memory

There are 16384 words of local memory associated with each background processor. Each word is 64 bits in length. This memory is treated as a register file to hold scalar or vector operands during a computation period and then returns the data to the common memory.

3.1.5 Instruction Stack

Each background processor has an instruction stack to allow program loops to execute without additional common memory references. This instruction stack consists of eight independent fields of 64 parcels of instruction data each. Programs may loop within these fields using any of the branch instructions.

3.1.6 Instruction Issue Control

Instruction parcels are positioned in a queue while awaiting the proper conditions for beginning execution. The parcels arrive at the queue from the instruction stack. They work their way up the queue to a location referred to as the issue position. The instruction parcel in the issue position must wait for the proper issue conditions as specified in the descriptions of the individual instructions. When the issue conditions are satisfied, the execution of the instruction begins and the next instruction moves to the issue position.

3.1.7 Floating-Point Multiply Unit

Each background processor has a floating-point multiply unit. This unit may accept operand data each clock period and, after a transit time delay, deliver a result each clock period. The unit is reserved for the time of a vector stream during execution of vector multiply instructions. The unit may accept scalar references as fast as they issue if the unit is not processing vector data.

3.1.8 floating-point Add Unit

Each background processor has a floating-point add unit. This unit may accept operand data each clock period and, after a transit time delay, deliver a result each clock period. The unit is reserved for the time of a vector stream during execution of vector addition instructions. The unit may accept scalar references as fast as they issue if the unit is not processing vector data.

3.1.9 Vector Integer Unit

Each background processor has a vector integer add unit. This unit may accept operand data each clock period and, after a transit time delay, deliver a result each clock period. The unit is reserved for the time of a vector stream during execution of vector integer instructions. This unit is not used for scalar computation.

3.1.10 Scalar Integer Unit

Each background processor has a scalar integer add unit. This unit may accept scalar operands as fast as the instructions can issue. This unit is not used for vector streams.

3.1.11 Vector Logical Unit

Each background processor has a vector logical unit. This unit may accept operand data each clock period and, after a transit time delay, deliver a result each clock period. The unit is reserved for the time of a vector stream during execution of vector logical instructions. This unit is not used for scalar computation. The RTC and vector mask registers are part of this functional unit.

3.1.12 Scalar Logical Unit

Each background processor has a scalar logical unit. This unit may accept scalar operands as fast as the instructions can issue. This unit is not used for vector streams.

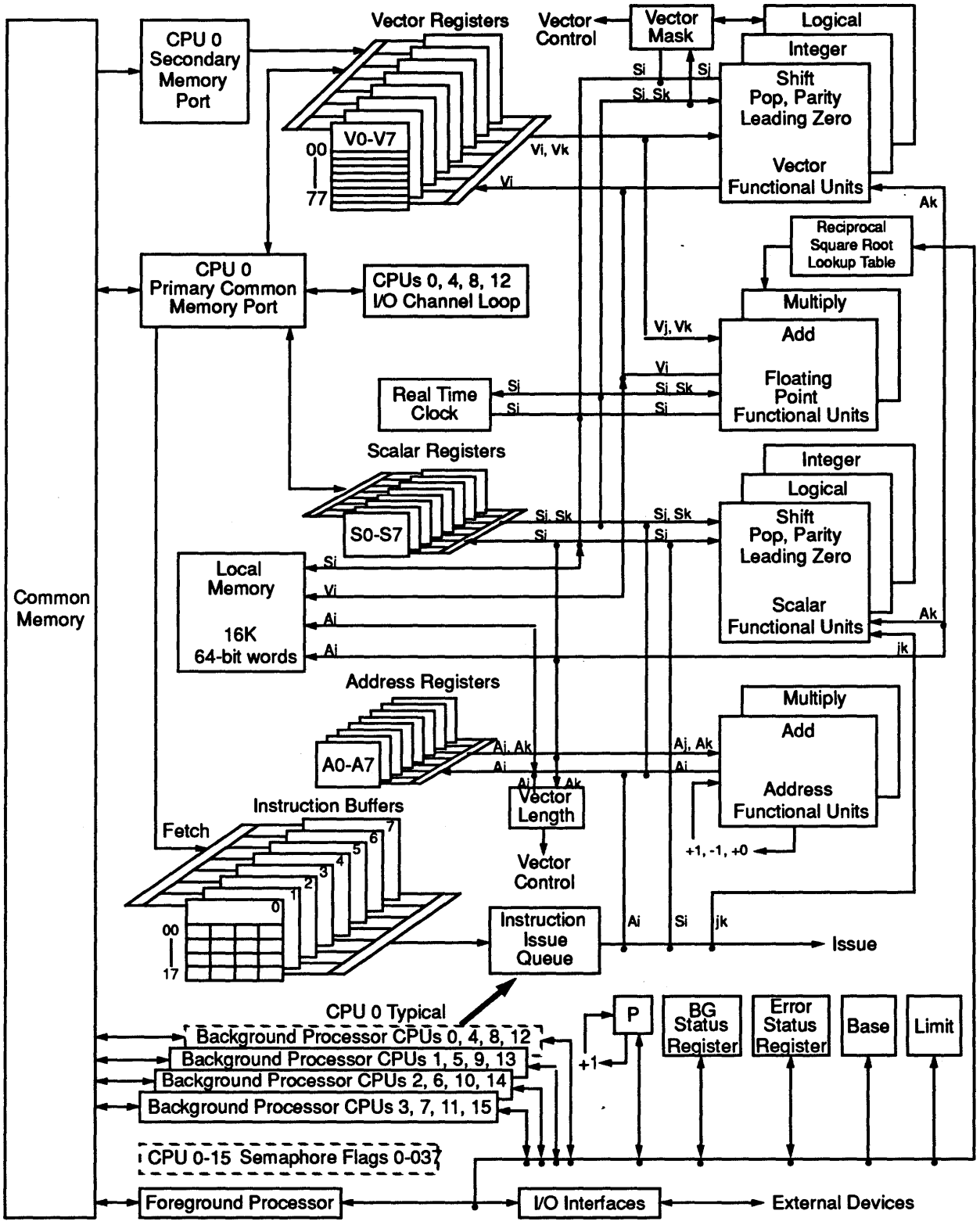
3.1.13 Address Add Unit

Each background processor has an address add unit for 32 bit integer addition.

3.1.14 Address Multiply Unit

Each background processor has an address multiply unit for 32 bit integer multiplication.

Figure 3 Background Processor Block Diagram



3.1.15 Local Memory Access

The local memory functions in a manner similar to a functional unit. The local memory is reserved during the time of a vector stream for those instructions which move data between local memory and the vector registers. The local memory is reserved for a few clock periods during address and scalar references. The delay due to a scalar reference depends on the character of the individual instruction.

3.1.16 Common Memory Access

Each background processor contains a read and write access path to common memory for scalar register references. Both paths are reserved during a scalar transfer, forcing transfers to occur one at a time. A single- or dual-port access is available for vector references. Utilizing the single port, common memory read or write references will reserve both paths. Dual-port access implements a read and a write path. This will allow a write reference to a vector register to issue immediately after a read reference to a different vector register. The background processor instruction summary contains information on instructions that utilize the single- or dual-port common memory access.

3.1.17 Vector Shift, Pop, LZC Unit

Each background processor has a vector shift unit which includes the functions of shift, population count and leading zero count. This unit may accept operand data each clock period and, after a transit time delay, deliver a result each clock period. The unit is reserved for the time of a vector stream during execution of vector shift instructions. This unit is not used for scalar computation.

3.1.18 Scalar Shift, Pop, LZC Unit

Each background processor has a scalar shift unit which includes the functions of shift, population count and leading zero count. This unit may accept scalar operands as fast as the instructions can issue. This unit is not used for vector streams.

3.1.19 Functional Units

Functional Unit	FU Timings (cp.) Scalar	Vector	Instructions
Address Add	3		020-021, 023
Address Multiply	12		022
Scalar Logical	1		100-103
Scalar Shift/Pop Parity	5/8		106-113
Scalar Integer	5		104, 105
Vector Logical		8	030-035, 114-115, 140-147
Vector Shift		11	150-153
Vector Integer		11	160-165, 176
Local Memory	5/6	10	044-047, 054-057, 074-077
Floating Add	12	16	120-123, 170-175
Floating Multiply (multiply) Floating Multiply (reciprocal)	13 17	17 21	124-127, 154-157 132, 133, 166, 167
Common Memory Read	35	40	001, 007, 060-073, 134, instruction fetch, I/O
Common Memory Write	35	40	001, 007, 060-073, 135, I/O

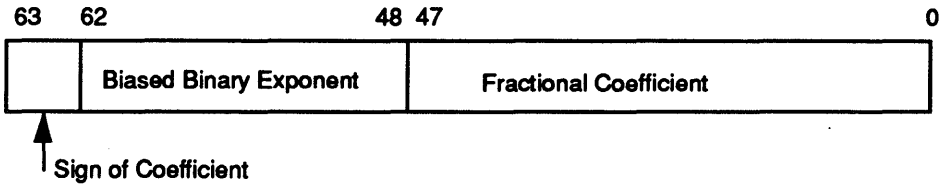
3.2 Floating-Point Data Format

Floating-point computation is performed on data in a special packed format within a 64 bit word. This format is generally referred to as 'sign and magnitude' to distinguish it from other forms in common use. The 64 bits of the data word are structured into three fields, as listed on the following page from highest order to lowest order bit position.

The numerical value of the floating-point data is determined by multiplying the coefficient times a base 2 with a signed binary exponent.

3.2.1 Coefficient

The 48 bit coefficient is a binary number with the binary point to the left of the highest order bit position. The highest order bit in the coefficient is normally a



1 bit - Sign of coefficient
15 bits - Biased binary exponent
48 bits - Fractional coefficient

one value. The floating-point number is said to be normalized if this is the case. Correct results in some functional units depend on normalized operands. The range of decimal values for the coefficient in a normalized form is 0.5 to 1.0.

3.2.2 Exponent

The 15 bit exponent is a binary number with a bias to allow integer testing of floating-point data. A value of 40000_8 represents a zero exponent. A larger value represents a positive exponent, and a smaller value represents a negative exponent. A floating-point representation of the integers zero, plus one, and minus one in a normalized form are then as follows in an octal form for each of the three fields.

Zero: 0 00000 0000000000000000

Plus one: 0 40001 4000000000000000

Minus one: 1 40001 4000000000000000

Exponent values of 60000_8 and greater are considered to have overflowed the exponent range. Hardware tests are performed for these values to indicate floating-point range error. Exponent values less than 20000_8 are considered to have underflowed the floating-point range. Such values are treated in many cases as if they had a zero value. There is no hardware indication when a computation underflows the floating-point range.

This floating-point format allows the expression of decimal numbers in the approximate range of 10 to the power -2466 through 10 to the +2466.

3.3 Fixed-Point Data Format

Fixed-point computation is performed on data in a twos complement integer format. Long-word arithmetic is performed with 64 bit integers. A fixed-point

representation of the integers zero, plus one, and minus one in this format are illustrated below using octal notation.

Zero: 00000000000000000000

Plus one: 00000000000000000001

Minus one: 17777777777777777777

Address computation and short integer computation are also performed in a twos complement integer format. In this case the arithmetic is performed with 32 bit integers. A fixed-point representation of the integers zero, plus one, and minus one in this format are illustrated below using octal notation.

Zero: 00000000000

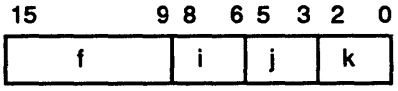
Plus one: 00000000001

Minus one: 37777777777

3.4 Instruction Format

The background processors instruction codes are represented by 16 bit parcels of data. These parcels are packed four per word in the common memory. The parcels are addressed as if the common memory had four times as many locations and the data were 16 bits long. A branch instruction to a parcel location out of the buffer area of the instruction stack results in a common memory reference. In this case the common memory address is formed in the instruction fetch hardware by shifting the instruction parcel address down by two bit positions.

Instructions are one, two, three or five parcels long. Instruction translation involves the interpretation of four designators within the first 16 bit parcel of the instruction. The basic format of the first parcel follows.



7 bits - f designator
3 bits - i designator
3 bits - j designator
3 bits - k designator

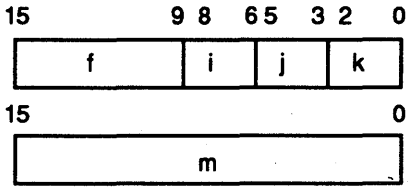
Single-Parcel Instruction Format

The f designator determines the instruction type, the length and the format of the remainder of the instruction. The f designator values are indicated in octal

notation in the instruction descriptions. The i, j, and k designators generally refer to vector, scalar, or address registers in a three-address format. The i designator generally specifies the destination register for the functional computation. The j and k designators generally specify the source operands.

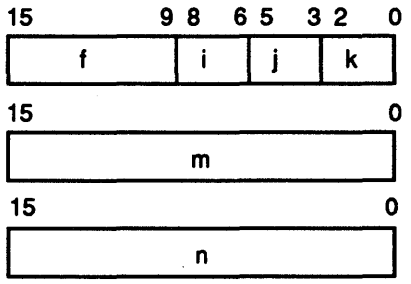
Constants are entered into the operating registers from the instruction stream. These constant values appear in parcels following the instruction which refers to them. There may be one, two or four parcels of constant data, depending upon the specific instruction. The first parcel in a multi-parcel group is always the highest order portion of the resulting constant.

Single-parcel constants are used to address the local memory. Two-parcel constants are used to address the common memory. Four-parcel constants are used to enter long word values into the scalar registers. Formats for two-, three- and five-parcel instructions follow.



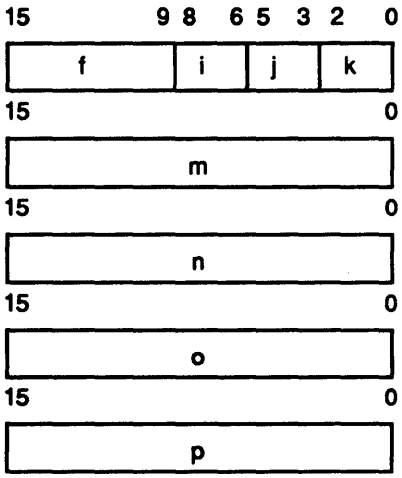
7 bits - f designator
3 bits - i designator
3 bits - j designator
3 bits - k designator
16 bits - m data

Two-Parcel Instruction Format



7 bits - f designator
3 bits - i designator
3 bits - j designator
3 bits - k designator
16 bits - m data
16 bits - n data

Three-Parcel Instruction Format



7 bits - f designator
3 bits - i designator
3 bits - j designator
3 bits - k designator
16 bits - m data
16 bits - n data
16 bits - o data
16 bits - p data

Five-Parcel Instruction Format

3.5 Vector (V) Registers

There are eight vector registers in each background processor. Each register contains 64 words of 64 bit length. These registers are used as a computational way station for data between the memory and the functional units.

The instruction issue control mechanism creates a read reservation and a write reservation for vector registers. When a vector register is selected as a destination register (V_i), the write reservation is set. Subsequent vector instructions utilizing the same vector register as a destination (V_i) or source (V_j or V_k) will not issue until the vector register is released. When a vector register is selected as a source (V_j or V_k), the read reservation is set. Later instructions that utilize the same vector register as a destination register (V_i) will be allowed to issue immediately. This is the tailgating feature provided by having separate read and write pointers for data leaving and entering each vector register. Subsequent vector instructions utilizing the same vector register as a source (V_j or V_k) will not issue until the read reservation is released. This method of vector register reservation eliminates hold issue conditions when

consecutive vector instructions are issued that read from and write to the same vector register.

The length of the vector stream is determined by the content of the vector length register. This register is set before the vector instruction issues and may be used for a number of vector operations of the same length. Maximum vector length is 64 elements. If vector length value is set to zero, all vector instructions except those that set the vector mask register (030 through 033) will execute as no-ops. The vector mask register instruction summary explains this special case.

The i, j, and k designators in a vector instruction may have the same value. In the case of identical source operands the data is streamed from the same vector register to both data paths. In the case of a source register, which is the same as a destination register, the data is sent to the functional unit, and the register will then be entered with result data from the functional unit.

The vector registers are implemented in the hardware with 64 by 4 register chips. These integrated circuit chips have a one clock cycle time.

The 64 elements of a vector register have independent read and write address registers. Consecutive element addresses for vector data advance the address pointer allowing data to stream into or out of a vector register in consecutive clock periods. Individual data elements originating from common memory during a vector register read may return out of order due to common memory conflicts. Background processor hardware guarantees that the data elements are returned to the correct position within the vector register.

3.6 Scalar (S) Registers

There are eight scalar registers in a background processor. Each register contains a single 64 bit word. These registers are used to support the vector registers in vector streaming where one element of the computation is a constant value. The scalar registers are used as computational way stations between the memory and the functional units where vector implementation of the work is not possible.

The instruction issue control mechanism reserves a scalar register which is the destination register for a functional operation. This interlocks subsequent instructions which wish to use the result in this register as a source operand. scalar registers used as the source for functional unit data are not reserved. The issue control mechanism tests the reservation flag at the time of issue for the instruction. If the register is free the data is immediately read and is free for the next instruction. Data is held at the functional unit for those cases where a scalar operand is used in a vector streaming operation.

The eight scalar registers as a group have several data read-out paths and one data destination path. Each path may be used independently in each clock period. An instruction requiring one or two scalar source operands reads these operands in clock phase five following instruction issue. The destination path must be reserved at issue time for the specific clock phase of result data arrival.

3.7 Address (A) Registers

There are eight Address (A) registers in a background processor. Each register contains a single 32 bit word. These registers are used to calculate memory locations for both the local memory and the common memory. They are also used to compute integer results in a 32 bit mode.

The instruction issue control mechanism reserves an address register which is the destination register for a functional operation. This interlocks subsequent instructions which wish to use the result in this register as a source operand. Address registers used as the source for functional unit data are not reserved. The issue control mechanism tests the reservation flag at the time of issue for the instruction. If the register is free the data is immediately read and is free for the next instruction. Data is held at the functional unit for those cases where an address operand is used in a vector streaming operation.

The eight address registers as a group have several data read-out paths and one data destination path. Each path may be used independently in each clock period. An instruction requiring one or two address source operands reads them during the clock period following instruction issue. The destination path must be reserved at issue time for the specific clock phase of result data arrival.

3.8 Local Memory

There are 16384 words of local memory associated with each background processor. Each word is 64 bits in length. This memory is treated as a register file to hold scalar or address operands during a computation period and then return the data to the common memory. The local memory may also be used for temporary storage of vector segments where these segments are used more than once in a computation in the vector registers.

The local memory is organized into four banks of 4096 words each. Each bank has its own address register. All four banks share a common data read-out register, but each bank has its own data write register. A scalar read reference initiates all four banks at the same clock phase. The proper bank is then sampled to the read-out register four clock periods later.

Parity is kept for each word in the local memory. It is generated when the word is written and checked when the word is read. Parity errors are reported in the background processor status register.

Vector references to the local memory require one extra clock period for address setup. This is to allow an arbitrary starting point in local memory for the vector stream. The bank address registers then advance sequentially for the length of the vector stream. Data moves at the rate of one word per clock period in vector mode.

The local memory is reserved at instruction issue time in the same manner as a functional unit. The reservation has a five-clock-period duration for a scalar or address reference. For a vector write to local memory or read from local memory the reservation lasts for 10 clock periods for vector lengths of 1, 2, 3 or 4 and for one additional clock period for each element beyond 4.

3.9 Instruction Stack

Each background processor has an instruction stack to allow program loops to execute without additional common memory references. This instruction stack has eight independent fields of 64 parcels of instruction data each. Programs may loop within the stack using any of the branch instructions.

The instruction stack is implemented in hardware with 64-by-4 register chips. These integrated circuit chips have a one-clock-period cycle time. A read or a write may take place each clock period.

Parity is kept for each word in the instruction stack. It is generated when the word is written and checked when the word is read. Parity errors are reported in the background processor status register and error data register.

The instruction stack is organized into two banks of 64 words each. Each word is 64 bits in length. Each bank has its own address register. The two banks share a common write data register and a read-out data register. Consecutive addresses in the instruction stack alternate between the two banks. The two independent banks allow common memory data to enter the instruction stack with a 64 bit wide word the same clock period that an instruction word is being read from the other bank not referenced by the common memory write.

The eight fields of data in the instruction stack cross the two banks of circuit packages. Each bank contains eight words, or 32 parcels of data, for each field. The common memory address for each field is held in one of eight field reference address registers. A branch instruction requires a test of the reference addresses to determine if the new common memory address is already represented in the instruction stack. This test is performed simultaneously on

all eight common memory addresses in the reference address registers. If the new instruction data is already in the instruction stack the read-out proceeds directly. If the new instruction data must come from common memory a fetch sequence is initiated.

A branch out of the instruction stack discards the oldest data field and replaces this field with a new 16 word field. A field boundary is anticipated 12 instruction parcels prior to the end of the current field.If the next sequential field is not in the stack, a common memory fetch is initiated. The next desired instruction word address is used as the reference address of the new field. The next 16 sequential addresses are then loaded into the buffer field. This mechanism permits the eight buffer fields to overlap at times. In the case of multiple coincidence tests the highest order field number is used for the pointer.

Each instruction word in the instruction stack has a word-valid tag associated with it. Each instruction field has a field-valid register associated with it. When an instruction fetch is initiated to an instruction field, the field-valid register for that field is toggled. As instruction words arrive from common memory, the word-valid tag is set equal to the field-valid register to indicate that the data is valid. As instruction words are read from the field to the instruction issue circuits, the word-valid tag is compared with the field-valid register to ensure that the instruction data is current.

The independent nature of the instruction fields makes it possible to have several concurrent fetches in progress, eliminating the need to wait for a fetch to complete before branching or crossing field boundaries.

3.10 Instruction Issue

Background instructions are translated in several steps and are allowed to issue sequentially by an instruction issue control mechanism. Instruction words are delivered one at a time from the instruction stack. These words are disassembled into parcels, which are placed in a queue where the translation occurs. The instruction issue process involves checking the reservation flags for the registers and functional unit involved in the instruction sequence. The parcel waits in the issue position in the queue until all required resources are free.

An instruction is considered to issue during the even phase of the last clock period in which it resides in the instruction issue queue. That instruction parcel is then discarded, and the next instruction parcel shifts forward in the queue. The instruction stack continues to fill the queue as the parcels shift forward.

The instruction issue mechanism makes reservations for the assigned resources during the clock period that the instruction issues. Tests for the next instruction can then be made against the updated reservation flags. This reservation and test process takes two clock phases (one clock period) for a complete interactive cycle. The maximum instruction issue rate is then one instruction every clock period.

Constants are intermixed with instruction parcels in the instruction stream. The constant parcels are passed through the instruction issue queue without test. Instructions containing parcels of constant data will delay issue of the next instruction by one clock period for each parcel of constant data.

Detail control signals are sent by the issue control mechanism to the register and functional unit circuitry in the two clock periods following issue. These signals are formed and transmitted concurrent with the translation and issue test of the following instruction parcel.

The instruction translation and issue control mechanism reserves and sends detail control signals to the destination register control at a later time. This delay corresponds to the transit time of the source operands through the functional unit. A delay chain in the instruction translation circuitry provides the memory vehicle for these destination tags.

The scalar registers have a single destination path for data entry into the eight member registers. The issue control mechanism must reserve this path during the appropriate clock phase in addition to the more obvious register reservation at issue time. The path reservation is for the one clock phase of intended use.

The address registers have a single destination path for data entry into the eight member registers. This path must be reserved for an address register destination in the same manner as for the scalar registers.

After an in-stack branch, instruction issue may be delayed for up to six phases (three clocks) waiting for the instruction issue pipe to fill with the necessary instruction/data parcels. This delay is incurred when the jump is to a non-word boundary address (i.e., parcels B, C, D) and the first one to four instructions at the branch address issue without any hold/delay conditions. The phase delay is computed as 2*parcel offset, so jumps to addresses ending in parcels A, B, C and D may experience a delay in instruction issue of zero, two, four or six phases, respectively.

For optimal performance it is recommended that the beginning of loops are aligned on word boundaries.

3.11 Floating-point Multiply Unit

The floating-point multiply unit performs floating-point calculations for both scalar and vector instructions. The unit is reserved for the period that vector elements are being streamed to it when a vector instruction is executed. Scalar use is locked out during this period. No reservation of the unit is made for a scalar floating-point multiply instruction. Successive scalar instructions may issue at the maximum rate.

The floating-point multiply unit accepts vector element pairs at the rate of one pair per clock period. The source operands move through the unit, stopping at an internal register briefly for each clock phase boundary. The transit time through the unit is a constant for each mode of operation.

3.11.1 Floating-point Product

The floating-point multiply unit forms the product of two operands in floating-point format and delivers the results in floating-point format. If both operands are normalized the result will also be normalized. The instructions that perform a floating-point multiply are:

Machine Code	CAL Instruction		Description
124ijk	S_i	S_j*FS_k	Floating-point product of S_j and S_k
125ijk	S_i	S_j*FS_k	Same as instruction 124
154ijk	V_i	S_j*FV_k	Floating-point products of S_j and V_k
155ijk	V_i	V_j*FV_k	Floating-point products of V_j and V_k

The product is formed using a bit-pair recoding scheme to merge the 48 bit coefficients into a reduced matrix of logical product circuits. All recoded product bits are formed in the same clock period and entered into a pyramid of 3 bit adder circuits. The exponents are summed to form the result exponent. A one-count decrement to the final exponent is required if the resulting coefficient must be shifted left one bit position for normalization.

The coefficient pyramid produces a 96 bit internal result. Since the final coefficient may require a normalization shift, two rounded 48 bit coefficients are derived. If the final coefficient does not need to be shifted for normalization, the rounded coefficient with a round into bit position -49 is chosen. If the final coefficient requires a shift for normalization, the coefficient with a round bit into bit position -50 will be shifted and selected.

During scalar/vector operations (154 instruction) the scalar operand is held in the floating multiply unit. This frees the scalar register for other use during the vector streaming period.

Several special cases are treated in the floating-point multiply unit for operands out of range:

1. One or both operands have overflow exponent (Exception: see case 5).
2. Result has overflow exponent.
3. Result has underflow exponent.
4. Both operand exponents are zero.
5. Either but not both operand exponents are zero.

Cases 1 and 2 will set a floating-point range error (bit 16) in the background processor status register and will set the result exponent to an overflow value of 60001_8. Cases 3 and 5 result in an all-zero word sent to the destination register. Case 4 computes the coefficient with no correction for normalization. This case aids multiple precision integer calculations.

3.11.2 Reciprocal Approximation

Reciprocal approximation is performed in two distinct steps. In the first step a first approximation is formed and one iteration is completed. This step proceeds in two parts: first a table lookup using read only memory, and then the terms from the lookup table are used to complete the first iteration of producing an improved approximation.

The second step is to perform another iteration, producing the reciprocal, and multiply the numerator by it to produce the quotient.

The instructions that perform a first reciprocal approximation are:

Machine Code	CAL Instruction		Description
132ij-	S_i	/HS_j	Reciprocal approximation of S_j
166ij-	V_i	/HV_j	Reciprocal approximation of V_j

The table used in the first step of the reciprocal approximation contains 2048 words, each 39 bits long. The address for the table lookup is taken from the highest 12 bits of the operand coefficient. The operand is assumed to be normalized so only 11 bits are used as the table address. These values are then used to complete the first iteration. See Appendix A for a detailed description of the reciprocal approximation sequence.

Three special cases are recognized in the reciprocal first approximation instructions:

1. Operand has overflow exponent.
2. Result has underflow exponent.
3. Result has overflow exponent.

All three cases will set a floating-point range error (bit 16) in the background processor status register, and the result exponent will be forced to an overflow value of 60002_8.

3.11.3 Reciprocal Iteration

The second iteration is implemented as a sequence of code that computes a correction term from the original operand and then applies it to the first approximation to produce the second approximation. The correction term is applied by performing a floating-point multiply. The final step of the sequence is to perform a floating-point multiply of the numerator by the reciprocal to produce the quotient.

The instructions that produce the reciprocal approximation correction term are:

Machine Code	CAL Instruction		Description
126ijk	S_i	S_j*IS_k	Reciprocal iteration 2-S_j*S_k
156ijk	V_i	V_j*IV_k	Reciprocal iteration 2-V_j*V_k

It is assumed that these instructions will only be used during a complete reciprocal sequence. Because the correction factor range is so narrow (approximately 1.0), the operand exponents are ignored, the result exponent is initialized to a value of 40001_8 before normalization, and the operand coefficients are multiplied. Since the operand exponents are ignored, there are no special range cases recognized during the iteration instructions. See Appendix A for a detailed description of the reciprocal approximation sequence.

3.11.4 Reciprocal Square Root Approximation

Square root approximation is performed in two distinct steps. In the first step a first approximation is formed and one iteration is completed. This step proceeds in two parts: first a table lookup using read only memory, and then the terms from the lookup table are used to complete the first iteration of producing an improved approximation.

The second step performs another iteration to produce the reciprocal square root and multiplies by the original operand to produce the square root.

The instructions that perform a first reciprocal square root approximation are:

Machine Code	CAL Instruction		Description
133ij-	S_i	$*QS_j$	Square root approximation of S_j
167ij-	V_i	$*QV_j$	Square root approximation of V_j

The table used in the first part of square root approximation contains 2048 words, each 39 bits long. The address of the table lookup is taken from the least significant bit of the exponent of the operand and the 11 most significant bits of the coefficient of the operand. Of these 11 bits of the coefficient, the first bit is discarded as the coefficient is assumed to be normalized. These values are used to complete the first iteration of producing an improved approximation. See Appendix A for a detailed description of the reciprocal square root approximation sequence.

There are five special cases recognized in the reciprocal square root approximation instructions:

1. Operand has an exponent value of 00000_8 through 00003_8.
2. Operand has overflow exponent.
3. Operand is a negative value (sign bit set).
4. Result has underflow exponent.
5. Result has overflow exponent.

Cases 2 through 5 will set a floating-point range error (bit 16) in the background processor status register; case 1 will not set the error flag. This prevents a range error for the square root of zero. All five cases will set the result exponent to an overflow value of 60004_8.

3.11.5 Reciprocal Square Root Iteration

The second iteration is implemented as a sequence that computes the second iteration and multiplies the reciprocal square root by the original operand to produce the square root. The iteration term is applied by performing a floating-point multiply after the first approximation has been multiplied by the original operand.

The instructions that produce the reciprocal square root approximation iteration term are:

Machine Code	CAL Instruction		Description
127ijk	S_i	S_j*QS_k	Square root iteration (3-S_j*S_k)/2
157ijk	V_i	V_j*QV_k	Square root iteration (3-V_j*V_k)/2

It is assumed that these instructions will only be used during a complete reciprocal square root sequence. Because the correction factor range is so narrow (approximately 1.0), the operand exponents are ignored, the result exponent is initialized to a value of 40001_8 before normalization, and the operand coefficients are multiplied. Since the operand exponents are ignored, there are no special range cases recognized during the iteration instructions. See Appendix A for a detailed description of the reciprocal square root approximation sequence.

3.12 Floating-point Add Unit

The floating-point add unit performs floating-point calculations and conversions between fixed-point and floating-point formats for both scalar and vector instructions. The unit is reserved for the period that vector elements are being streamed to it when a vector instruction is executed. Scalar use is locked out during this period. No reservation of the unit is made for a scalar floating-point add instruction. Successive scalar instructions may issue at the maximum rate.

The floating-point add unit accepts vector element operands at the rate of one pair per clock period. The source operands move through the unit, stopping at an internal register briefly for each clock phase boundary. The transit time through the unit is a constant for all modes of operation. scalar register destination results will leave the floating-point add unit after 12 clock periods and vector results will leave four clock periods later.

3.12.1 Floating-point Addition

The floating-point add unit forms the sum of two operands in floating-point format and produces a result in floating-point format. The result will always be

normalized regardless of whether the source operands were normalized. The instructions that perform a floating-point addition are:

Machine Code	CAL Instruction		Description
120ijk	S_i	S_j+FS_k	Floating-point sum of S_j and S_k
170ijk	V_i	S_j+FV_k	Floating-point sums of S_j and V_k
171ijk	V_i	V_j+FV_k	Floating-point sums of V_j and V_k

3.12.2 Floating-point Subtraction

The floating-point add unit forms the difference of two floating-point operands and produces a result in floating-point format. This mode of operation is essentially identical to the addition mode. The only difference is the toggle of the sign bit for the second operand. The instructions that perform a floating-point subtraction are:

Machine Code	CAL Instruction		Description
121ijk	S_i	S_j-FS_k	Floating-point difference of S_j and S_k
172ijk	V_i	S_j-FV_k	Floating-point differences of S_j and V_k
173ijk	V_i	V_j-FV_k	Floating-point differences of V_j and V_k

3.12.3 Floating-point to Fixed-point Conversion

The floating-point add unit forms a fixed-point representation of a floating-point operand. This process is accomplished by adding the operand to a constant in floating-point format and then masking the coefficient field into the fixed-point result. The result must be complemented and a 1 added if the result is negative. This mechanism already exists in the mid-sequence option of the floating-point addition sequence. The instructions that perform floating-point to fixed-point format conversion are:

Machine Code	CAL Instruction		Description
122i-k	S_i	FIX, S_k	Convert S_k to integer
174i-k	V_i	FIX, V_k	Convert V_k to integers

3.12.4 Fixed-point to Floating-point Conversion

The floating-point add unit forms a floating-point representation of a fixed-point format operand. This process is accomplished by adding the operand to a constant and using the floating-point normalization hardware to form the proper floating-point result. The instructions that perform fixed-point to floating-point format conversion are:

Machine Code	CAL Instruction		Description
123i-k	S_i	FLT, S_k	Convert S_k to floating-point
175i-k	V_i	FLT, V_k	Convert V_k to floating-point

3.12.5 Operation of the Floating-point Add Unit

The floating-point add unit has an addition and a subtraction mode to satisfy all the possible combinations of operands and functions it is required to perform. The instruction type and the sign bits of the operands determine the functional unit mode.

Add mode:

- The instruction is an addition and the sign bits of the operands are the same.
- The instruction is a subtraction and the signs of the operands are different.
- The instruction is a conversion of a positive number.

Subtract mode:

- The instruction is a subtraction and the sign bits of the operands are the same
- The instruction is an addition and the sign bits of the operands are different.
- The instruction is a conversion of a negative number.

The unit performs several steps as it executes an instruction:

1. The Exponent Adder
2. The Coefficient Shift
3. The Coefficient Adder
4. Coefficient Normalization
5. Sign bit calculation.

3.12.5.1 Exponent Adder

The first step of the adder is to subtract the biased exponents. This accomplishes two goals:

- It determines the shift required to align the two coefficients.
- It determines the approximate exponent of the result.

This subtraction is performed by adding the exponent of the first operand with the complement of the exponent of the second operand as two 14 bit ones complement integers. The difference between the exponents is called the 'shift count'. If the result of this operation is contained in a 14 bit result, then the approximate exponent of the result with be that of the first operand. If the result of this operation exceeds the 14 bit result, then the exponent of the second operand is used for the approximate exponent of the result.

3.12.5.2 Coefficient Shift

The coefficients of the operands are aligned using the shift count generated by the exponent adder to produce two operands of the same magnitude. The operand with the smallest exponent is shifted by the shift count to be of the same magnitude as the operand with the larger exponent.

If the unit is in subtract mode, the coefficient of the operand with the larger exponent (the operand that is not being shifted) is complemented.

3.12.5.3 Coefficient Adder

The shifted and unshifted coefficients enter a 48 bit adder. The least significant bit position is formed by the adder from three bits: the shift bit, the unshifted bit, and a round bit. The round bit is set if the last bit that was shifted off the shift unit was a one. The round bit is disabled for a floating-point to fixed-point format conversion or if the shift count exceeds 48.

The result of the addition is complemented for two cases when the unit is in subtract mode:

- The addition does not produce a carry-out of the 48 bit adder.
- The result is the conversion of a negative number.

3.12.5.4 Coefficient Normalization

When the 48 bit result of the coefficient adder does not have a 1 bit in the most significant position the result is not normalized. There are two cases that cause the 48 bit result from the coefficient adder not to be a normalized number:

- In subtract mode the result has a number of leading zeros.

- In addition mode the result has a carry-out of the 48 bit adder.

In the first case the number of leading zeros is used as a shift count to shift the coefficient to the left so that a 1 bit is set in the most significant bit position. The approximate exponent is also decremented by this shift count.

In the second case the 48 bit result coefficient is shifted one position to the right, the most significant bit of the result exponent is set, and the approximate exponent is incremented by one.

These normalization strategies are disabled during the conversion from fixed-point to floating-point format. During this step the result is set to zero if the biased exponent is less than 20000_8. or if trying to convert an integer of more than 48 bits is attempted. This attempt to make an integer greater than 48 bits is called 'constant overflow'.

3.12.5.5 Sign Bit Generation

The sign bits of the two operands follow the same selection path as the exponents of the operands, and the final sign is selected along with the exponent. The final sign bit is cleared in the following cases:

- Exponent overflow (the biased exponent is greater than 60000_8).
- Exponent underflow (the biased exponent is less than 20000_8).
- The leading zero count in the normalization operation is 48 and there is not a carry-out of the 48 bit coefficient adder.
- Constant overflow in the normalization operation during a fixed-point to floating-point conversion.

The sign bit is toggled for the following reasons:

- A carry-out of the 48 bit adder occurred in the coefficient adder when the unit is in subtract mode.
- Conversion of negative numbers during conversion instructions.

3.12.5.6 Floating-point to Fixed-point Conversion

Conversion from floating-point to fixed-point format is performed by adding a floating-point constant provided by the unit as the second operand (4006000000000000000_8) to the operand supplied by the instruction. This addition causes the coefficient of the operand being converted to be shifted into the 48 bit integer field of the result. If the operand being converted is negative, the result is complemented and a one is added to produce a two's complement representation of the negative integer. The unit performs the following steps:

1. The unit performs the normal alignment process, but if the exponent of the operand being converted is greater than 40060_8, then the resulting integer would be greater than 48 bits, and a constant overflow is generated and a floating-point range error (bit 16) is set in the background processor status register.
2. The coefficient shift performs normally.
3. The coefficient adder adds the coefficient of the operand being converted and zero (the coefficient of the supplied constant).
4. Coefficient normalization is disabled.
5. Final exponent adjustment is used to sign extend the 48 bit integer.

3.12.5.7 Fixed-point to Floating-point Conversion

Conversion from fixed-point to floating-point format is performed by adding a constant floating-point operand supplied by the unit as the second operand (4006000000000000000_8) to the operand being converted and normalizing the result. The unit performs the following steps:

1. The exponent alignment process is disabled.
2. The coefficient shift is disabled.
3. The coefficient add or subtract process operates normally.
4. Coefficient normalization performs normally.
5. Exponent adjustment performs normally.

3.12.5.8 Underflow and Overflow

An underflow condition is generated when the biased exponent of the result is less than 20000_8 or the coefficient of the result is all zeroes. Both conditions produce a result of all zeroes, but no range error is generated.

An overflow condition is generated if an attempt is made to create an integer longer than 48 bits during floating-point format to fixed-point format conversion. The exponent and coefficient will be cleared. An overflow condition also exists if the biased exponent of the result exceeds 60000_8. In both cases a floating-point range error (bit 16) is set in the background processor status register.

3.13 Vector Mask (VM) Register

The vector mask register is a 64 bit special purpose register which is implicitly referenced in the background processor instructions. The vector mask register is used to merge vector data according to a set of precomputed element flags. In effect it provides a vehicle for vectorizing conditional operations.

One bit of the vector mask register is associated with each element in the 64 element vector registers. The highest order bit of the vector mask corresponds to the first element of the vector data. The bits of the mask then proceed in order to represent the rest of the vector's elements.

The vector mask data can be formed by a vector streaming operation in which each element of the stream is evaluated for a specific criterion. The instructions for this purpose are the following:

Machine Code	CAL Instruction		Description
030--k	VM	V_k, Z	Set VM from zero elements of V_k
031--k	VM	V_k, N	Set VM from nonzero elements of V_k
032--k	VM	V_k, P	Set VM from positive elements of V_k
033--k	VM	V_k, M	Set VM from negative elements of V_k

The vector mask register is cleared at the beginning of these instruction sequences, and then bits are entered one at a time as the vector stream passes the test station.

The vector mask data can be used to merge two vector streams into a single result stream. The instructions for this purpose are the following:

Machine Code	CAL Instruction		Description
146ijk	V_i	S_j!V_k&VM	Merge S_j (VM=1) and V_k (VM=0)
147ijk	V_i	V_j!V_k&VM	Merge V_j (VM=1) and V_k (VM=0)

Elements of the j operand are selected where there are one bits in the mask. Elements of the k operand are selected where there are zero bits in the mask.

Data may be moved to or from the vector mask register using an scalar register as a way station. Instructions for this purpose are the following:

Machine Code	CAL Instruction		Description
034-j-	VM	S_j	Copy S_j to VM
114i--	S_i	VM	Copy VM to S_i

3.14 Vector Length (VL) Register

The vector length register is a seven bit special purpose register which is implicitly referenced in the background processor instructions. The vector length register is used to hold the vector segment length during a portion of the background computation. All vector streaming operations capture the segment length at the time of instruction issue from the vector length register.

The allowed values of vector length are 1 through 64. A zero value is interpreted as a vector length of zero, in which case all vector instructions except those that set the vector mask register (030 through 033) will pass through one clock period at a time before issuing as a no-op. Values larger than 64 are interpreted as 64. If the value of VL is less than 64, the elements beyond vector length in the destination register are unaffected. The vector length register may be set by entering the lowest order seven bits of A_k.

The instructions which communicate explicitly with the vector length register are the following:

Machine Code	CAL Instruction		Description
025i--	A_i	VL	Copy VL to A_i
036--k	VL	A_k	Copy A_k to VL

3.15 Real Time Clock (RTC)

Each background processor has a 64 bit register which counts continuously at the clock period rate. This count value may then be used to determine the passage of real time to an accuracy of one clock period. The instructions which communicate directly with the real time count register are the following:

Machine Code	CAL Instruction		Description
035-j-	RT	S_j	Copy S_j to RT (Monitor Mode only)
115i--	S_i	RT	Copy real time count to S_i

The lowest order two bits of the Real Time Clock (RTC) are forced to zeros when the clock is loaded from an scalar register by the 035 instruction.

The RTC in all 16 background processors are synchronized to zero at system deadstart.

3.16 Base Address (BA) Register

Each background processor has a base address register which defines the lower boundary of the common memory address field. The base address register is 32 bits in length. Data is entered into this register from the foreground processor while the background processor is in an idle mode. It remains in this register for the duration of the background processor computation period.

Each common memory reference from the background processor includes the addition of the base address register content to the other components of the memory reference address. All background references to common memory are thus relative to the base address boundary. The absolute common memory address is formed in the interface between the background processor and the common memory access port. Program instruction references as well as data references are treated in this manner.

3.17 Limit Address (LA) Register

Each background processor has a limit address register which defines the upper boundary of the common memory address field. The limit address register is 32 bits in length. Data is entered into this register from the foreground processor while the background processor is in an idle mode. It remains in this register for the duration of the background processor computation period.

Each common memory reference from the background processor includes a test of the resulting absolute common memory address against the content of the limit address register. A common memory range error indicator (bit 13 or bit 14) is set in the background processor status register if the resulting absolute common memory address is equal to, or is greater than, the limit address. A read reference results in zero data for this case. A write reference is aborted.

3.18 Program (P) Address Register

Each background processor has a program address register which indicates the program instruction parcel address for the parcel currently in the issue position during normal operation. The program addressregister is 32 bits in length. Data is entered from the foreground processor at the beginning of a computation period. The program address register content is advanced by one count as each parcel issues from the instruction issue queue.

The program address register content is reset to the branch destination address when a jump instruction is executed in the program sequence. The program address register content may be read by the foreground processor at any time during background processor operation.

3.19 Semaphore Flags

There are 32 semaphore flags in the background system used to interlock common memory references when multiple background processors are executing a single job. One semaphore flag is assigned to each job which is currently active in the background system. A background processor assigned to a job is then assigned a semaphore flag at the same time. This assignment is made by a five bit pointer in the background processor status register. The background processor status register semaphore pointer field is updated via a unique exit request from the background processor to the foreground. Once the semaphore request is complete, all subsequent semaphore manipulation instructions in the requesting background processor operate on the semaphore bit pointed to by the background processor status register pointer field.

There are four instructions the background processor uses in synchronizing its common memory references. These are as follows:

Machine Code	CAL Instruction		Description
004---	JCS	mn	Jump if semaphore is clear and set it
005---	JSS	mn	Jump if semaphore is set and set it
006---	SSM		Set semaphore
007---	CSM		Clear semaphore when memory quiet

The semaphore flag is requested by an 004 or 005 instruction when the background processor program wishes access to a common memory area which may interfere with other processors assigned to this job. The branch instruction results determine when the processor has exclusive access to this common memory area. The program clears the semaphore flag to release the common memory area to another processor assigned to the same job.

3.20 Background Processor Instruction Descriptions

The following descriptions for the background processor instructions include information to aid in timing studies. Decoding of an instruction and checking the availability of source registers, destination register and functional unit for its execution are overlapped with the execution of the previous instruction. An instruction is considered to issue in the first clock period it is in issue position, and there are no delaying conditions. Completion times indicate the number of clock phases before the resource can be used again. For functional units the time is the number of clock phases before another instruction using the same functional unit is allowed to issue. For destination registers time is the number of clock phases before an instruction utilizing the register as either a source or destination register is allowed to issue. Completion times are given for source registers only if they are not available for use when the next instruction is in issue position.

Some instructions read constants from the following parcels in the instruction stream. In such cases the program address is advanced over these data parcels to point to the next instruction. The highest order data parcel is read first for those cases of multi-parcel data.

The instruction descriptions begin with the octal code for the highest order seven bits of the parcel (the f field). The three octal register designators (the i, j and k fields) then follow. A dash appears in the description where a register's designator is ignored.

Vector registers always begin a read or write operation with the zero element position in the vector register. Elements are read or written sequentially for the length of the current vector data, which is stored in the vector length register. A short vector after a long vector leaves the old vector data in those positions not replaced with new data.

3.21 Background Processor Instruction Summary

The legend for the following list is explained below. Functional units and their acronyms are as follows:

Functional Unit	Acronym
Address Adder	AA
Address Multiply	AM

Functional Unit	Acronym
Address Adder	AA
Scalar Logical	SL
Scalar Shift	SS
Scalar Integer	SI
Vector Logical	VL
Vector Shift	VS
Vector Integer	VI
Local Memory	LM
Floating Add	FA
Floating Multiply	FM
Common Memory	CM

Machine Code	CAL Instruction		Func Unit	Description
000000	ERR			Error exit (EXIT 00)
000-jk	EXIT	jk		Normal exit
001---	CMR			Complete memory references
002ij-	R,A_i	A_j		Jump to A_j with return address in A_i
003---mn	J	mn		Unconditional jump
004---mn	JCS	mn		Jump if semaphore is clear and set it
005---mn	JSS	mn		Jump if semaphore is set and set it
006---	SSM			Set semaphore
007---	CSM			Clear semaphore when memory is quiet
010-j-mn	JZ	A_j,mn		Jump if A_j is zero
011-j-mn	JN	A_j,mn		Jump if A_j is nonzero
012-j-mn	JP	A_j,mn		Jump if A_j is positive or zero
013-j-mn	JM	A_j,mn		Jump if A_j is negative
014-j-mn	JZ	S_j,mn		Jump if S_j is zero
015-j-mn	JN	S_j,mn		Jump if S_j is nonzero
016-j-mn	JP	S_j,mn		Jump if S_j is positive or zero
017-j-mn	JM	S_j,mn		Jump if S_j is negative

Machine Code	CAL Instruction		Func Unit	Description
020ijk	A_i	A_j+A_k	AA	Integer sum of A_j and A_k
021ijk	A_i	A_j-A_k	AA	Integer difference of A_j and A_k
022ijk	A_i	A_j*A_k	AM	Integer product of A_j and A_k
023ij0	A_i	A_j	AA	Copy A_j to A_i
023ij1	A_i	A_j+1	AA	Increment A_j
023ij2	A_i	A_j-1	AA	Decrement A_j
024ij-	A_i	S_j		Copy S_j to A_i
025i--	A_i	VL		Copy VL to A_i
026ijk	A_i	jk,S,P		Load 6 bit positive value
027ijk	A_i	jk,S,M		Load 6 bit negative value
030--k	VM	V_k,Z	VL	Set VM from zero elements of V_k
031--k	VM	V_k,N	VL	Set VM from nonzero elements of V_k
032--k	VM	V_k,P	VL	Set VM from positive elements of V_k
033--k	VM	V_k,M	VL	Set VM from negative elements of V_k
034-j-	VM	S_j	VL	Copy S_j to VM
035-j-	RT	S_j	VL	Copy S_j to RT (Monitor Mode only)
036--k	VL	A_k		Copy A_k to VL
037--0	DRI			Disable halt on memory range error
037--1	ERI			Enable halt on memory range error
037--2	DFI			Disable halt on floating-point error
037--3	EFI			Enable halt on floating-point error
040i--m	A_i	m,P,P		Load A_i with 16 bit value and zero fill
041i--m	A_i	m,P,M		Load A_i with 16 bit value and ones fill
042i--mn	A_i	mn,H		Load A_i with 32 bit value
043i--mn	A_i	mn,H		Same as instruction 042
044i--m	A_i	[m]	LM	Direct read A_i from LM
045-j-m	[m]	A_j	LM	Write sign extended A_j to LM
046i-k	A_i	$[A_k]$	LM	Read A_i from LM

Machine Code	CAL Instruction		Func Unit	Description
047-jk	$[A_k]$	A_j	LM	Write sign extended A_j to LM
050i--mn	S_i	mn,H,P		Load lower half S_i; zero fill
051i--mn	S_i	mn,H,M		Load lower half S_i; ones fill
052i--mn	S_i	mn,L		Load upper half S_i; zero fill
053i--mnop	S_i	mnop,F		Load S_i with 64 bit value
054i--m	S_i	[m]	LM	Direct read S_i from LM
055i--m	[m]	S_i	LM	Direct write S_i to LM
056i-k	S_i	$[A_k]$	LM	Indirect read S_i from LM
057i-k	$[A_k]$	S_i	LM	Indirect write S_i to LM
060ijk	S_i	(A_j,A_k)		Read S_i from CM (A_j+A_k)
061ijk	(A_j,A_k)	S_i		Write S_i to CM (A_j+A_k)
062i-k	S_i	(A_k)		Read S_i from CM (A_k)
063i-k	(A_k)	S_i		Write S_i to CM (A_k)
064i-kmn	S_i	(A_k,mn)		Read S_i from CM (A_k+mn)
065i-kmn	(A_k,mn)	S_i		Write S_i to CM (A_k+mn)
066i--mn	S_i	(mn)		Read S_i from CM (mn)
067i--mn	(mn)	S_i		Write S_i to CM (mn)
070ijk	V_i	(A_j,A_k)		One-port read V_i from CM (A_j) stride A_k
071ijk	(A_j,A_k)	V_i		One-port write V_i to CM (A_j) stride A_k
072ijk	V_i	(A_k,V_j)		One-port gather V_i from CM (A_k+V_j)
073ijk	(A_k,V_j)	V_i		One-port scatter V_i to CM (A_k+V_j)
074i--m	V_i	[m]	LM	Direct read V_i from LM
075i--m	[m]	V_i	LM	Direct write V_i to LM
076i-k	V_i	$[A_k]$	LM	Indirect read V_i from LM (A_k)
077i-k	$[A_k]$	V_i	LM	Indirect write V_i to LM (A_k)
100ijk	S_i	$S_j\&S_k$	SL	Logical product of S_j and S_k
101ijk	S_i	$\#S_k\&S_j$	SL	Logical product of S_j and NOT S_k
102ijk	S_i	$S_j\backslash S_k$	SL	Logical difference of S_j and S_k
103ijk	S_i	$S_j!S_k$	SL	Logical sum of S_j and S_k

Machine Code	CAL Instruction		Func Unit	Description
104ijk	S_i	S_j+S_k	SI	Integer sum of S_j and S_k
105ijk	S_i	S_j-S_k	SI	Integer difference of S_j and S_k
106ij0	S_i	PS_j	SS	Population count of S_j
106ij1	S_i	QS_j	SS	Population count parity of S_j
107ij-	S_i	ZS_j	SS	Leading zero count of S_j
110ijk	S_i	$S_i<jk$	SS	Shift S_i left jk places
111ijk	S_i	$S_i>jk$	SS	Shift S_i right jk places
112ijk	S_i	$S_i,S_j<A_k$	SS	Shift S_i and S_j left A_k places
113ijk	S_i	$S_j,S_i>A_k$	SS	Shift S_i and S_j right A_k places
114i--	S_i	VM	VL	Copy VM to S_i
115i--	S_i	RT	VL	Copy real time count to S_i
116ijk	S_i	jk,S,P		Load 6 bit positive value
117ijk	S_i	jk,S,M		Load 6 bit negative value
120ijk	S_i	S_j+FS_k	FA	Floating-point sum of S_j and S_k
121ijk	S_i	S_j-FS_k	FA	Floating-point difference S_j and S_k
122i-k	S_i	FIX,S_k	FA	Convert S_k to integer
123i-k	S_i	FLT,S_k	FA	Convert S_k to floating-point
124ijk	S_i	S_j*FS_k	FM	Floating-point product of S_j and S_k
125ijk	S_i	S_j*FS_k	FM	Same as instruction 124
126ijk	S_i	S_j*IS_k	FM	Reciprocal iteration $2-S_j*S_k$
127ijk	S_i	S_j*QS_k	FM	Square root iteration $(3-S_j*S_k)/2$
130ij-	S_i	A_j		Copy A_j to S_i
131ij-	S_i	$+A_j$		Copy A_j to S_i with sign extension
132ij-	S_i	$/HS_j$	FM	Reciprocal approximation of S_j
133ij-	S_i	$*QS_j$	FM	Square root approximation of S_j
134ijk	V_i	$<A_j,A_k>$		Two-port read V_i from CM (A_j) stride A_k
135ijk	$<A_j,A_k>$	V_i		Two-port write V_i to CM (A_j) stride A_k
136ijk				Reserved for future use
137ijk				Reserved for future use
140ijk	V_i	$S_j\&V_k$	VL	Logical products of S_j and V_k

Machine Code	CAL Instruction		Func Unit	Description
141ijk	V_i	$V_j\&V_k$	VL	Logical products of V_j and V_k
142ijk	V_i	$S_j\backslash V_k$	VL	Logical differences of S_j and V_k
143ijk	V_i	$V_j\backslash V_k$	VL	Logical differences of V_j and V_k
144ijk	V_i	$S_j!V_k$	VL	Logical sums of S_j and V_k
145ijk	V_i	$V_j!V_k$	VL	Logical sums of V_j and V_k
146ijk	V_i	$S_j!V_k\&VM$	VL	Merge S_j (VM=1) and V_k (VM =0)
147ijk	V_i	$V_j!V_k\&VM$	VL	Merge V_j (VM =1) and V_k (VM=0)
150ijk	V_i	$V_j<A_k$	VS	Shift V_j left A_k bits
151ijk	V_i	$V_j>A_k$	VS	Shift V_j right A_k bits
152ijk	V_i	$V_j,V_j<A_k$	VS	Continuous shift V_j left A_k places
153ijk	V_i	$V_j,V_j>A_k$	VS	Continuous shift V_j right A_k places
154ijk	V_i	S_j*FV_k	FM	Floating-point products of S_j and V_k
155ijk	V_i	V_j*FV_k	FM	Floating-point products of V_j and V_k
156ijk	V_i	V_j*IV_k	FM	Reciprocal iteration $2-V_j*V_k$
157ijk	V_i	V_j*QV_k	FM	Square root iteration $(3-V_j*V_k)/2$
160ijk	V_i	S_j+V_k	VI	Integer sums of S_j and V_k
161ijk	V_i	V_j+V_k	VI	Integer sums of V_j and V_k
162ijk	V_i	S_j-V_k	VI	Integer differences of S_j and V_k
163ijk	V_i	V_j-V_k	VI	Integer differences of V_j and V_k
164ij0	V_i	PV_j	VS	Population counts of V_j
164ij1	V_i	QV_j	VS	Population count parities of V_j
165ij-	V_i	ZV_j	VS	Leading zero counts of V_j
166ij-	V_i	$/HV_j$	FM	Reciprocal approximations of V_j
167ij-	V_i	$*QV_j$	FM	Square root approximations of V_j
170ijk	V_i	S_j+FV_k	FA	Floating-point sums of S_j and V_k
171ijk	V_i	V_j+FV_k	FA	Floating-point sums of V_j and V_k
172ijk	V_i	S_j-FV_k	FA	Floating-point differences S_j and V_k
173ijk	V_i	V_j-FV_k	FA	Floating-point differences of V_j and V_k
174i-k	V_i	FIX,V_k	FA	Convert V_k to integers

Machine Code	CAL Instruction		Func Unit	Description
175i-k	V_i	FLT,V_k	FA	Convert V_k to floating-point
176ijk	V_i	CI,S_j&S_k	VI	Compressed iota of mask S_j and stride S_k
177---	PASS			No-op

Instruction 000

Machine Code	CAL Instruction	Description
000-jk	EXIT jk	Normal exit

This instruction terminates the current program sequence and places the background processor (BP) into an idle mode. The exit mode flag and idle mode flag are set in bits 6 and 22, respectively, of the BP status register. The six-bit quantity jk is copied from the instruction and entered into the BP status register bits 0 through 5.

Issue Conditions:

- None

Completion Times:

- All memory references complete
- Foreground response

Instruction 001

Machine Code	CAL Instruction	Description
001---	CMR	Complete memory references

This instruction issues as a null function and causes a delay in the issue of the following instruction until all previous CM references have cleared the memory port.

Issue Conditions:

- None

Completion Times:

- Memory port quiet
- Next instruction in issue position - phase 20

Instruction 002

Machine Code	CAL Instruction	Description
002ij-	R,A_i A_j	Jump to A_j with return address in A_i

This instruction terminates the current program sequence and begins a new sequence at a computed parcel address. The parcel address is read from A_j. The parcel address for the next instruction in the current program sequence is entered into A_i.

Issue Conditions:

- A_i free in phase 3
- A_j free in phase 3

Completion Times:

- A_i free in phase 11
- Next instruction in issue position as follows:
- Branch in stack - phase 34
- Branch out of stack - phase 120†
- There may be a delay of up to six phases after an in-stack branch waiting for the instruction pipe to fill with the next parcel

† Time is for no delay in the Instruction Stack Fetch Process.

Instruction 003

Machine Code	CAL Instruction		Description
003---	J	mn	Unconditional jump

This instruction terminates the current program sequence and begins a new sequence at a specified constant parcel address. The parcel address is read from the following two parcels of the instruction stream.

Issue Conditions:

- The following two parcels are in position in the issue queue

Completion Times:

- Next instruction in issue position as follows:
- Branch in stack - phase 34
- Branch out of stack - phase 118†
- There may be a delay of up to six phases after an in-stack branch waiting for the instruction pipe to fill with the next parcel

† Time is for no delay in the Instruction Stack Fetch Process.

Instructions 004 & 005

Machine Code	CAL Instruction		Description
004---	JCS	mn	Jump if semaphore is clear and set it
005---	JSS	mn	Jump if semaphore is set and set it

These two instructions conditionally terminate the current instruction sequence and begin a new sequence at a specified constant parcel address. The parcel address is read from the following two parcels of the instruction stream.

The branch is conditional on the state of the semaphore flag assigned to this processor. The semaphore flag is set for either instruction if it was not previously set. The semaphore flag bit in the BP status register (bit 28) is set if this instruction alters the state of the flag from a zero to a one value.

Issue Conditions:

The following two parcels are in position in the issue queue

Completion Times:

- Next instruction in issue position as follows:
- (There is an 18 to 50 phase delay for the test of the semaphore bit before the following times occur.)
- Branch fall through - phase 14
- Branch in stack - phase 34
- Branch out of stack - phase 124†
- There may be a delay of up to six phases after an in-stack branch waiting for the instruction pipe to fill with the next parcel

† Time is for no delay in the Instruction Stack Fetch Process.

Instruction 006

Machine Code	CAL Instruction	Description
006---	SSM	Set semaphore

This instruction sets the semaphore flag assigned to this BP without regard to its previous state. The semaphore flag bit in the BP status register (bit 28) is set if the previous state of the semaphore flag was a zero. This instruction is used by the operating system program to restore semaphore flag values at the time of job restart.

Issue Conditions:

- None

Completion Times:

- After an 18 to 50 phase delay for the test of the semaphore bit the next instruction will be in issue position in phase 2.

Instruction 007

Machine Code	CAL Instruction	Description
007---	CSM	Clear semaphore when memory quiet

This instruction clears the semaphore flag assigned to this BP without regard to its previous value. The semaphore flag bit in the BP status register (bit 28) is cleared by the execution of this instruction. This instruction is used by a BP program to release access to a privileged area of CM for other processors assigned to this job.

This instruction issues without delay. Execution of the function may be delayed by activity in the CM access port. The following instruction will not issue until the CM port is quiet. The semaphore flag will not clear until the CM port is quiet. This delay is necessary to ensure that any CM write operations have been completed before another processor is allowed access to the privileged area.

Issue Conditions:

- None

Completion Times:

- After an 18 to 50 phase delay for the test of the semaphore bit the next instruction will be in issue position in phase 2.

Instructions 010 - 013

Machine Code	CAL Instruction		Description
010-j-	JZ	A_j,mn	Jump if A_j is zero
011-j-	JN	A_j,mn	Jump if A_j is nonzero
012-j-	JP	A_j,mn	Jump if A_j is positive or zero
013-j-	JM	A_j,mn	Jump if A_j is negative

These instructions conditionally terminate the current instruction sequence and begin a new sequence at a specified constant parcel address. The parcel address is read from the following two parcels of the instruction stream.

The branch is conditional on the content of A_j as indicated above. The current program sequence is continued if the branch criterion is not met.

Issue Conditions:

- A_j free in phase 3
- The following two parcels are in position in the issue queue

Completion Times:

- Next instruction in issue position as follows:
- Branch fall through - phase 16
- Branch in stack - phase 34
- Branch out of stack - phase 118†
- There may be a delay of up to six phases after an in-stack branch waiting for the instruction pipe to fill with the next parcel

† Time is for no delay in the Instruction Stack Fetch Process.

Instructions 014 - 017

Machine Code	CAL Instruction		Description
014-j-	JZ	S_j,mn	Jump if S_j is zero
015-j-	JN	S_j,mn	Jump if S_j is nonzero
016-j-	JP	S_j,mn	Jump if S_j is positive or zero
017-j-	JM	S_j,mn	Jump if S_j is negative

These instructions conditionally terminate the current instruction sequence and begin a new sequence at a specified constant parcel address. The parcel address is read from the following two parcels of the instruction stream.

The branch is conditional on the content of S_j as indicated above. The current program sequence is continued if the branch criterion is not met.

Issue Conditions:

- S_j free in phase 5
- The following two parcels are in position in the issue queue

Completion Times:

- Next instruction in issue position as follows:
- Branch fall through - phase 18
- Branch in stack - phase 34
- Branch out of stack - phase 118†
- There may be a delay of up to six phases after an in-stack branch waiting for the instruction pipe to fill with the next parcel

† Time is for no delay in the Instruction Stack Fetch Process.

Instruction 020

Machine Code	CAL Instruction		Description
020ijk	A_i	A_j+A_k	Integer sum of A_j and A_k

This instruction forms the 32 bit integer sum of two 32 bit integer operands. The operands are obtained from A_j and A_k. The result is delivered to A_i.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- A_i free in phase 3
- Address register destination path free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- Address add unit free in phase 2
- A_i free in phase 9

Instruction 021

Machine Code	CAL Instruction		Description
021ijk	A_i	A_j-A_k	Integer difference of A_j and A_k

This instruction forms the 32 bit integer difference of two 32 bit integer operands. The operands are obtained from A_j and A_k. The result is delivered to A_i.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- A_i free in phase 3
- Address register destination path free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- A_i free in phase 9
- Address add unit free in phase 2

Instruction 022

Machine Code	CAL Instruction		Description
022ijk	A_i	A_j*A_k	Integer product of A_j and A_k

This instruction forms the integer product of two 32 bit integer operands. The operands are obtained from A_j and A_k. The lower 32 bits of the result data are delivered to A_i.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- A_i free in phase 3
- Address register destination path free in phase 26
- Issue will be held for two phases if another 022 instruction immediately precedes this one

Completion Times:

- Next instruction in issue position in phase 2
- Address multiply unit free in four phases
- A_i free in phase 27

Instruction 023

Machine Code	CAL Instruction		Description
023ij0	A_i	A_j	Copy A_j to A_i
023ij1	A_i	A_j+1	Increment A_j
023ij2	A_i	A_j-1	Decrement A_j

This instruction forms the 32 bit integer sum of a 32 bit integer operand and a small constant from the instruction stream. The 32 bit integer operand comes from A_j. The small constant has values 0, +1 or -1, depending upon the value of k as follows: (see note)

For	k = 0	increment value = 0
	k = 1	+1
	k = 2	-1

- It is not advisable to use k values greater than 2, as they may be assigned specific meanings at a later time. k values other than those specified above will also affect the increment value as follows:

For	k = 3, 4 or 7	increment value = 0
	k = 5	+1
	k = 6	-1

Issue Conditions:

- A_i free in phase 3
- A_j free in phase 3
- Address register destination path free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- Address add unit free in two phases
- A_i free in phase 9

Instruction 024

Machine Code	CAL Instruction		Description
024ij-	A_i	S_j	Copy $S_{j \text{ to } Ai}$

This instruction reads a 64 bit word from S_j and enters the lower order 32 bits into A_i.

Issue Conditions:

- A_i free in phase 3
- S_j free in phase 5
- Address register destination path free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- A_i free in phase 9

Instruction 025

Machine Code	CAL Instruction	Description
025i--	A_i VL	Copy VL to A_i

This instruction forms a 32 bit word from the data in the vector length register. The low order seven bits are copied from the corresponding bits of the vector length data. The higher order bits are zero. The resultant data is delivered into A_i.

Issue Conditions:

- A_i free in phase 3
- Address register destination path free in phase 6

Completion Times:

- Next instruction in issue position in phase 2
- A_i free in phase 7

Instruction 026

Machine Code	CAL Instruction		Description
026ijk	A_i	jk,S,P	Load 6 bit positive value

This instruction forms a 32 bit word from the jk data in the instruction parcel. The low order six bits are copied from the instruction parcel. The higher order bits are zero. The result data is delivered into A_i.

Issue Conditions:

- A_i free in phase 3
- Address register destination path free in phase 6

Completion Times:

- Next instruction in issue position in phase 2
- A_i free in phase 7

Instruction 027

Machine Code	CAL Instruction		Description
027ijk	A_i	jk,S,M	Load 6 bit negative value

This instruction forms a 32 bit word from the jk data in the instruction parcel. The low order six bits are copied from the instruction parcel. The higher order bits are ones. The result data is delivered to A_i.

Issue Conditions:

- A_i free in phase 3
- Address register destination path free in phase 6

Completion Times:

- Next instruction in issue position in phase 2
- A_i free in phase 7

Instructions 030 - 033

Machine Code	CAL Instruction		Description
030--k	VM	V_k,Z	Set VM from zero elements of V_k
031--k	VM	V_k,N	Set VM from nonzero elements of V_k
032--k	VM	V_k,P	Set VM from positive elements of V_k
033--k	VM	V_k,M	Set VM from negative elements of V_k

These instructions stream the V_k data to the vector logical unit. The VM register is initially cleared. A bit is then entered into the VM register where elements of the vector stream meet the test criterion. The highest order bit position in the VM register corresponds to the first element of the vector. The bit positions are then assigned in order for the remainder of the vector stream. If the vector length is 0, the VM register is cleared.

Issue Conditions:

- V_k free in phase 5
- VM register in Vector logical unit free in phase 10
- Vector logical unit free in phase 10

Completion Times:

- Next instruction in issue position in phase 2
- V_k register free in phase 17†
- VM register in Vector logical unit free in phase 12 if VL = 0
- VM register in Vector logical unit free in phase 20†
- Vector logical unit free in phase 20†

† This time is for vector length = 1, 2, 3 or 4. Add one clock period for each additional element beyond 4.

Instruction 034

Machine Code	CAL Instruction	Description
034-j-	VM S_j	Copy S_j to VM

This instruction enters the VM register with a 64 bit word from S_j.

Issue Conditions:

- S_j free in phase 5
- VM register in Vector logical unit free in phase 8
- Vector logical unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2 (phase 6 if a 114 instruction is next in the queue)
- VM register free in phase 11

Instruction 035

Machine Code	CAL Instruction		Description
035-j-	RT	S_j	Copy S_j to RT (Monitor Mode only)

This instruction sets the contents of the 64 bit RTC equal to the number in S_j. The RTC can be entered only if bit 21 of the BP status register has been set. Bits 0 and 1 of S_j are forced to zero in this instruction to insure that the RTC value is accurate.

Issue Conditions:

- S_j free in phase 5
- RTC in vector logical unit free in phase 8
- Vector logical unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2 (phase 6 if a 115 instruction is next in the queue)
- RTC free in phase 11

Instruction 036

Machine Code	CAL Instruction		Description
036--k	VL	A_k	Copy $A_{k\ to\ VL}$

This instruction enters the vector length register with data from A_k. The values entered depends on the value of A_k as follows:

$0 \leq A_k \leq 64$ VL is set to the value of A_k

$A_k < 0$; $A_k > 64$ VL is set to 64

Issue Conditions:

- A_k free in phase 3

Completion Times:

- Next instruction in issue position in phase 20

Instruction 037

Machine Code	CAL Instruction	Description
037--0	DRI	Disable halt on memory range error
037--1	ERI	Enable halt on memory range error
037--2	DFI	Disable floating-point range tests
037--3	EFI	Enable floating-point range tests

This instruction alters four status bits in the BP status register, depending upon the value of the k designator in the instruction parcel. The k designator values and related actions are as follows:

k = 0 Disable interrupt on CM range error (bit 15)

k = 1 Clear CM read range error (bit 13)

Clear CM write range error (bit 14)

Enable interrupt on CM range error (bit 15)

k = 2 Disable floating-point range tests (bit 18)

k = 3 Enable floating-point range tests (bit 18)

All CM and floating-point range errors will be reported to the status register, regardless of whether the corresponding Interrupt on range errors bits have been enabled.

The instructions for enabling or disabling of CM or floating-point range errors do not hold issue waiting for instructions that can generate range error conditions to complete. To ensure that error interrupts are correctly generated and reported, the programmer should insert a Complete Memory Reference (CMR) before either a DRI or ERI instruction or by using the result of a floating-point operation before either a DFI or EFI instruction.

Issue Conditions:

- None

Completion Times:

- Next instruction in issue position in phase 20

Instruction 040

Machine Code	CAL Instruction	Description
040i--	A_i m,P,P	Load A_i with 16 bit value, zero fill

This instruction enters A_i with a 32 bit constant. The lower order 16 bits are read from the following parcel of the instruction stream. The higher order 16 bits are zero filled.

Issue Conditions:

- A_i free in phase 3
- Address register destination path free in phase 6
- The following parcel is in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 4
- A_i free in phase 7

Instruction 041

Machine Code	CAL Instruction		Description
041i--	A_i	m,P,M	Load A_i with 16 bit value, ones fill

This instruction enters A_i with a 32 bit constant. The lower order 16 bits are read from the following parcel of the instruction stream. The higher order 16 bits are filled with ones.

Issue Conditions:

- A_i free in phase 3
- Address register destination path free in phase 6
- The following parcel is in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 4
- A_i free in phase 7

Instructions 042 & 043

Machine Code	CAL Instruction		Description
042i--mn	A_i	mn,H	Load A_i with 32 bit value
043i--mn	A_i	mn,H	Same as instruction 042

These instructions enter A_i with a 32 bit constant which is read from the following two parcels of the instruction stream.

Issue Conditions:

- A_i free in phase 3
- Address register destination path free in phase 8
- The following two parcels are in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 6
- A_i free in phase 9

Instruction 044

Machine Code	CAL Instruction		Description
044i--	A_i	[m]	Direct read A_i from Local Memory

This instruction enters A_i with the lower order 32 bits of a data word in local memory. The local memory address of the data word is obtained from the following parcel in the instruction stream.

Issue Conditions:

- A_i free in phase 3
- Local memory free in phase 9
- Address register destination path free in phase 20
- The following parcel is in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 4
- Local memory free in phase 21
- A_i free in phase 21

Instruction 045

Machine Code	CAL Instruction	Description
045-j-	[m] A_j	Write sign extended A_j to LM

This instruction stores one 64 bit word into the local memory. The local memory address is obtained from the following parcel in the instruction stream. The data word is obtained by sign extending the content of A_j through the higher order bit positions of the 64 bit word.

Issue Conditions:

- A_j free in phase 3
- Local memory free in phase 9
- The following parcel is in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 4
- Local memory free in phase 21

Instruction 046

Machine Code	CAL Instruction		Description
046i-k	A_i	$[A_k]$	Read A_i from Local Memory

This instruction enters A_i with the lower order 32 bits of a data word in local memory. The local memory address of the data word is obtained from A_k.

Issue Conditions:

- A_k free in phase 3
- A_i free in phase 3
- Local memory free in phase 9
- Address register destination path free in phase 20

Completion Times:

- Next instruction in issue position in phase 2
- Local memory free in phase 19
- A_i free in phase 21

Instruction 047

Machine Code	CAL Instruction	Description
047-jk	$[A_k]$ A_j	Write sign extended A_j to LM

This instruction stores one 64 bit word into the local memory. The local memory address is obtained from A_k. The write data word is obtained by sign extending the content of A_j through the higher order bit positions of the 64 bit word.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- Local memory free in phase 9

Completion Times:

- Next instruction in issue position in phase 2
- Local memory free in phase 19

Instruction 050

Machine Code	CAL Instruction	Description
050i--	S_i mn,H,P	Load lower half S_i; zero fill

This instruction enters S_i with a 64 bit constant. The lower order 32 bits are read from the following two parcels of the instruction stream. The higher order 32 bits are zero filled.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 10
- The following two parcels are in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 6
- S_i free in phase 11

Instruction 051

Machine Code	CAL Instruction	Description
051i--	S_i mn,H,M	Load lower half S_i; ones fill

This instruction enters S_i with a 64 bit constant. The low order 32 bits are obtained from the following two parcels in the instruction stream. The higher order 32 bits are filled with ones.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 10
- The following two parcels are in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 6
- S_i free in phase 11

Instruction 052

Machine Code	CAL Instruction		Description
052i--	S_i	mn,L	Load upper half S_i; zero fill

This instruction enters S_i with a 64 bit constant. The upper 32 bits of this constant are obtained from the following two parcels in the instruction stream. The lower order 32 bits are zero filled.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 10
- The following two parcels are in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 6
- S_i free in phase 11

Instruction 053

Machine Code	CAL Instruction	Description
053i--	S_i mnop,F	Load S_i with 64 bit constant

This instruction enters S_i with a 64 bit constant which is read from the following four parcels of the instruction stream.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 14
- A two-phase (one clock period) delay is required before the instruction is issued to guarantee the following four parcels are in position in the issue queue. This delay is reflected in the completion times.

Completion Times:

- Next instruction in issue position in phase 10
- S_i free in phase 15

Instruction 054

Machine Code	CAL Instruction	Description
054i--	S_i [m]	Direct read S_i from Local Memory

This instruction enters S_i with a 64 bit data word from the local memory. The local memory address is obtained from the following parcel in the instruction stream.

Issue Conditions:

- S_i free in phase 5
- Local memory free in phase 9
- Scalar register destination path free in phase 20
- The following parcel is in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 4
- Local memory free in phase 21
- S_i free in phase 21

Instruction 055

Machine Code	CAL Instruction	Description
055i--	[m] S_i	Direct write S_i to Local Memory

This instruction stores one 64 bit word into the local memory. The local memory address is obtained from the following parcel in the instruction stream. The data word is obtained from S_i.

Issue Conditions:

- S_i free in phase 5
- Local memory free in phase 9
- The following parcel is in position in the issue queue

Completion Times:

- Next instruction in issue position in phase 4
- Local memory free in phase 21

Instruction 056

Machine Code	CAL Instruction		Description
056i-k	S_i	$[A_k]$	Indirect read S_i from Local Memory

This instruction enters S_i with a 64 bit data word from the local memory. The local memory address is obtained from A_k.

Issue Conditions:

- A_k free in phase 3
- S_i free in phase 5
- Local memory free in phase 9
- Scalar register destination path free in phase 20

Completion Times:

- Next instruction in issue position in phase 2
- Local memory free in phase 19
- S_i free in phase 21

Instruction 057

Machine Code	CAL Instruction	Description
057i-k	[A_k] S_i	Indirect write S_i to Local Memory

This instruction stores one 64 bit word into the local memory. The local memory address is obtained from A_k. The data word is obtained from S_i.

Issue Conditions:

- A_k free in phase 3
- S_i free in phase 5
- Local memory free in phase 9

Completion Times:

- Next instruction in issue position in phase 2
- Local memory free in phase 19

Instruction 060

Machine Code	CAL Instruction		Description
060ijk	S_i	(A_j,A_k)	Read S_i from Common Memory (A_j+A_k)

This instruction reads one 64 bit word from CM and enters it into S_i. The CM address is formed by first adding the content of A_j to the content of A_k. This sum is then added to the BP base address to determine the CM address.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the scalar memory reference. This prevents subsequent scalar memory references from getting out of order in the event of a memory port backup.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- S_i free in phase 5
- Scalar register destination path free in phase 74
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 2
- S_i is free in phase 75. This completion time assumes no delays in the memory bank access.

Instruction 061

Machine Code	CAL Instruction	Description
061ijk	(A_j,A_k) S_i	Write S_i to Common Memory (A_j+A_k)

This instruction stores one 64 bit word from S_i into CM. The CM address is formed by first adding the content of A_j to the content of A_k. This sum is then added to the BP base address to determine the CM address.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the scalar memory reference. This prevents subsequent scalar memory references from getting out of order in the event of a memory port backup.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- S_i free in phase 5
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference.
- No memory port backup.

Completion Times:

- Next instruction in issue position in phase 2

Instruction 062

Machine Code	CAL Instruction	Description
062i-k	S_i (A_k)	Read S_i from Common Memory (A_k)

This instruction reads one 64 bit word from CM and enters it into S_i. The CM address is formed by adding the content of A_k to the BP base address.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the scalar memory reference. This prevents subsequent scalar memory references from getting out of order in the event of a memory port backup.

Issue Conditions:

- A_k free in phase 3
- S_i free in phase 5
- Scalar register destination path free in phase 74
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 2
- S_i is free in phase 75. This completion time assumes no delays in the memory bank access.

Instruction 063

Machine Code	CAL Instruction	Description
063i-k	(A_k) S_i	Write S_i to Common Memory (A_k)

This instruction stores one 64 bit word from S_i into CM. The CM address is formed by adding the content of A_k to the BP base address.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the scalar memory reference. This prevents subsequent scalar memory references from getting out of order in the event of a memory port backup.

Issue Conditions:

- A_k free in phase 3
- S_i free in phase 5
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 2

Instruction 064

Machine Code	CAL Instruction	Description
064i-k	S_i (A_k,mn)	Read S_i from Common Memory (A_k+mn)

This instruction reads one 64 bit word from CM and enters it into S_i. The CM address is formed by first adding the content of A_k to a 32 bit constant. This sum is then added to the BP base address to determine the CM address. The constant is obtained from the next two parcels of instruction stream data.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the scalar memory reference. This prevents subsequent scalar memory references from getting out of order in the event of a memory port backup.

Issue Conditions:

- A_k free in phase 3
- S_i free in phase 5
- Scalar register destination path free in phase 74
- The following two parcels are in position in the issue queue
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 6
- S_i is free in phase 75. This completion time assumes no delays in the memory bank access.

Instruction 065

Machine Code	CAL Instruction	Description
065i-k	(A_k,mn) S_i	Write S_i to Common Memory (A_k+mn)

This instruction stores one 64 bit word from S_i into CM. The CM address is formed by first adding the content of A_k to a 32 bit constant. This sum is then added to the BP base address to determine the CM address. The constant is obtained from the next two parcels of instruction stream data.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the scalar memory reference. This prevents subsequent scalar memory references from getting out of order in the event of a memory port backup.

Issue Conditions:

- A_k free in phase 3
- S_i free in phase 5
- The following two parcels are in position in the issue queue
- Memory port not active with a CM vector operation (70-73, 134, 135) instruction
- Memory port not active with fetch or foreground reference
- No memory port backup.

Completion Times:

- Next instruction in issue position in phase 6

Instruction 066

Machine Code	CAL Instruction	Description
066i--	S_i (mn)	Read S_i from Common Memory (mn)

This instruction reads one 64 bit word from CM and enters it into S_i. The CM address is formed by adding a 32 bit constant to the BP base address. The constant is obtained from the next two parcels of instruction stream data.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the scalar memory reference. This prevents subsequent scalar memory references from getting out of order in the event of a memory port backup.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 74
- The following two parcels are in position in the issue queue
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 6
- S_i is free in phase 75. This completion time assumes no delays in the memory bank access.

Instruction 067

Machine Code	CAL Instruction		Description
067i--	(mn)	S_i	Write S_i to Common Memory (mn)

This instruction stores one 64 bit word from S_i into CM. The CM address is formed by adding a 32 bit constant to the BP base address. The constant is obtained from the next two parcels of instruction stream data.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the scalar memory reference. This prevents subsequent scalar memory references from getting out of order in the event of a memory port backup.

Issue Conditions:

- S_i free in phase 5
- The following two parcels are in position in the issue queue
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 6

Instruction 070

Machine Code	CAL Instruction		Description
070ijk	V_i	(A_j,A_k)	One-port read V_i from CM (A_j) stride A_k

This instruction reads a vector stream of 64 bit words from CM into V_i. The length of the stream is determined by the content of the vector length register. This instruction issues immediately as a null function if the vector length is zero.

The first element address for the CM reference is formed by adding the content of A_j to the BP base address. The following addresses of the CM reference are separated by constant increments. The increment value is read from A_k.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the vector function.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- V_i free in phase 5
- No CM scalar (60-67) instruction issued in last 20 phases (10 clock periods)
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 2
- Memory port is inactive in 26 phases†. This completion time assumes no memory port backup occurs.
- V_i is free in phase 85†. This completion time assumes no delays in the memory bank access.

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

Instruction 071

Machine Code	CAL Instruction	Description
071ijk	(A_j,A_k) V_i	One-port write V_i to CM (A_j) stride A_k

This instruction stores a vector stream of 64 bit words into CM from V_i. The length of the stream is determined by the content of the vector length register. This instruction issues immediately as a null function if the vector length is zero.

The first element address for the CM reference is formed by adding the content of A_j to the BP base address. The following addresses of the CM reference are separated by constant increments. The increment value is read from A_k.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the vector function.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- V_i free in phase 5
- No CM scalar (60-67) instruction issued in last 20 phases (10 clock periods)
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 2
- Memory port is inactive in 26 phases†. This completion time assumes no memory port backup occurs.
- V_i is free in phase 17††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 072

Machine Code	CAL Instruction	Description
072ijk	V_i (A_k,V_j)	One-port gather V_i from CM (A_k+V_j)

This instruction reads a vector stream of 64 bit words from CM into V_i. The length of the stream is determined by the content of the vector length register. This instruction issues immediately as a null function if the vector length is zero.

Each CM element address is separately computed in the execution of this instruction. The content of A_k is read at the beginning of instruction execution and is held in the memory access port. The content of V_j is then streamed to the memory access port. The highest order 32 bits of this data is ignored. The lowest order 32 bits are used in the individual element address calculations.

The first result element address is formed by adding the first element of V_j data to the A_k register data. This sum is then added to the BP base address to determine the CM address for the first result element. The following result element addresses are calculated using subsequent V_j element values and repeatedly using the A_k register data and BP base address.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the vector function.

Issue Conditions:

- A_k free in phase 3
- V_i free in phase 5
- V_j free in phase 5
- No CM scalar (60-67) instruction issued in last 20 phases (10 clock periods)
- Memory port not active with a CM vector operation (70-73, 134, 135)
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 2
- Memory port is inactive in 26 phases (13 clock periods)†. This completion time assumes no memory port backup occurs.
- V_i is free in phase 87†. This completion time assumes no delays in the memory bank access.
- V_j is free in phase 17††.

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 073

Machine Code	CAL Instruction	Description
073ijk	(A_k,V_j) V_i	One-port scatter V_i to CM (A_k+V_j)

This instruction stores a vector stream of 64 bit words into CM from V_i. The length of the stream is determined by the content of the vector length register. This instruction issues immediately as a null function if the vector length is zero.

Each CM element address is separately computed in the execution of this instruction. The content of A_k is read at the beginning of instruction execution and is held in the memory access port. The content of V_j is then streamed to the memory access port. The highest order 32 bits of this data is ignored. The lowest order 32 bits are used in the individual element address calculations.

The first result element address is formed by adding the first element of V_j data to the A_k register data. This sum is then added to the BP base address to determine the CM address for the first result element. The following result element addresses are calculated using subsequent V_j element values and repeatedly using the A_k register data and BP base address.

The memory reference is made in a single-port mode. In this mode the read and write functions of the CM port are both reserved for the vector function.

Issue Conditions:

- A_k free in phase 3
- V_i free in phase 5
- V_j free in phase 5
- No CM scalar (60-67) instruction issued in last 20 phases (10 clock periods)
- Memory port not active with a CM vector operation (70-73, 134, 135).
- Memory port not active with fetch or foreground reference.
- No memory port backup.

Completion Times:

- Next instruction in issue position in phase 2
- Memory port is inactive in 26 phases (13 clock periods)†. This completion time assumes no memory port backup occurs.
- V_i is free in phase 17††.
- V_j is free in phase 17††.

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 074

Machine Code	CAL Instruction	Description
074i--	V_i [m]	Direct read V_i from Local Memory

This instruction reads a stream of 64 bit words from local memory at consecutive locations. The initial address for the local memory field is obtained from the following parcel in the instruction stream. The data stream is entered into V_i. The length of the stream is specified by the content of the vector length register.

Issue Conditions:

- V_i free in phase 5
- Local memory free in phase 9

Completion Times:

- Next instruction in issue position in phase 4
- V_i free in phase 29†
- Local memory free in phase 29††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 075

Machine Code	CAL Instruction	Description
075i--	[m] V_i	Direct write V_i to Local Memory

This instruction stores a stream of 64 bit words into local memory at consecutive locations. The initial address for the local memory field is obtained from the following parcel in the instruction stream. The data stream is obtained from V_i. The length of the stream is specified by the content of the vector length register.

Issue Conditions:

- V_i free in phase 5
- Local memory free in phase 9

Completion Times:

- Next instruction in issue position in phase 4
- V_i free for use as a source register in phase 17†
- Local memory free in phase 29†

† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 076

Machine Code	CAL Instruction		Description
076i-k	V_i	$[A_k]$	Indirect read V_i from Local Memory

This instruction reads a stream of 64 bit words from local memory at consecutive locations. The initial address for the local memory field is obtained from A_k. The data stream is entered into V_i. The length of the stream is specified by the content of the vector length register.

Issue Conditions:

- A_k free in phase 3
- V_i free in phase 5
- Local memory free in phase 9

Completion Times:

- Next instruction in issue position in phase 2
- V_i free in phase 29†
- Local memory free in phase 29††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 077

Machine Code	CAL Instruction	Description
077i-k	$[A_k]$ V_i	Indirect write V_i to Local Memory

This instruction stores a stream of 64 bit words into local memory at consecutive locations. The initial address for the local memory field is obtained from the following parcel in the instruction stream. The data stream is obtained from V_i. The length of the stream is specified by the content of the vector length register.

Issue Conditions:

- A_k free in phase 3
- V_i free in phase 5
- Local memory free in phase 9

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 17†
- Local memory free in phase 29†

† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 100

Machine Code	CAL Instruction		Description
100ijk	S_i	$S_j\&S_k$	Logical product of S_j and S_k

This instruction reads two 64 bit scalar operands, forms the bit-by-bit logical product and delivers the result to S_i. The operands are obtained from S_j and S_k.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- Scalar register destination path free in phase 6

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 7

Instruction 101

Machine Code	CAL Instruction		Description
101ijk	S_i	$\#S_k\&S_j$	Logical product of S_j and NOT S_k

This instruction reads two 64 bit scalar operands, forms a bit-by-bit logical product and delivers the result to S_i. The operands are obtained from S_j and S_k. The S_k data is inverted before the logical product is formed.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- Scalar register destination path free bit-by-bit logical difference phase 6

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 7

Instruction 102

Machine Code	CAL Instruction		Description
102ijk	S_i	$S_j \backslash S_k$	Logical difference of S_j and S_k

This instruction reads two 64 bit scalar operands, forms their bit-by-bit logical difference and delivers the result to S_i. The operands are obtained from S_j and S_k.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- Scalar register destination path free in phase 6

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 7

Instruction 103

Machine Code	CAL Instruction		Description
103ijk	S_i	$S_j!S_k$	Logical sum of S_j and S_k

This instruction reads two 64 bit scalar operands, forms a bit-by-bit logical sum and delivers the result to S_i. The operands are obtained from S_j and S_k.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- Scalar register destination path free in phase 6

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 7

Instruction 104

Machine Code	CAL Instruction		Description
104ijk	S_i	S_j+S_k	Integer sum of S_j and S_k

This instruction reads two 64 bit scalar operands, forms their integer sum and delivers the result to S_i. The operands are obtained from S_j and S_k.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- Scalar register destination path free in phase 14

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 15

Instruction 105

Machine Code	CAL Instruction		Description
105ijk	S_i	S_j-S_k	Integer difference of S_j and S_k

This instruction reads two 64 bit scalar operands, forms their integer difference and delivers the result to S_i. The minuend is obtained from S_j and the subtrahend from S_k.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- Scalar register destination path free in phase 14

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 15

Instruction 106

Machine Code	CAL Instruction		Description
106ij0	S_i	PS_j	Population count of S_j
106ij1	S_i	QS_j	Population count parity of S_j

This instruction reads a 64 bit operand from S_j and forms a count of the number of one bits in that operand. This count is delivered as a positive seven-bit result to S_i.

If the lowest order bit of the k designator is set, the seven-bit result from the population count is masked so that only the lowest order bit is delivered to the destination scalar register. This serves as an odd parity bit for S_j.

Issue Conditions:

- S_j free in phase 5
- S_i free in phase 5
- Scalar register destination path free in phase 20

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 21

Instruction 107

Machine Code	CAL Instruction	Description
107ij-	S_i ZS_j	Leading zero count of S_j

This instruction reads a 64 bit operand from S_j and forms a count of the number of leading zeros in the operand. The operand is considered as a field of 64 individual bits in this operation. The resulting count may have the values zero through 64. The result is delivered to S_i as a positive integer.

Issue Conditions:

- S_j free in phase 5
- S_i free in phase 5
- Scalar register destination path free in phase 20

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 21

Instruction 110

Machine Code	CAL Instruction		Description
110ijk	S_i	S_i<jk	Shift S_i left jk places

This instruction reads a 64 bit operand from S_i, shifts the data to the left and returns it to S_i. The number of bit positions in the shift count is a constant from the instruction parcel. This constant has a value equal to the lowest order six bits in the parcel (the j and k fields). The range of this constant is then zero through 63.

The data is shifted left in an open-ended manner. That is, zero bits are inserted from the right as bits shift off to the left.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 14

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 15

Instruction 111

Machine Code	CAL Instruction		Description
111ijk	S_i	S_i>jk	Shift S_i right jk places

This instruction reads a 64 bit operand from S_i, shifts the data to the right and returns it to S_i. The number of bit positions in the shift count is a constant from the instruction parcel. This constant has a value equal to the lowest order six bits in the parcel. The range of this constant is then zero through 63.

The data is shifted right in an open-ended manner. That is, zero bits are inserted from the left as bits shift off to the right.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 14

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 15

Instruction 112

Machine Code	CAL Instruction	Description
112ijk	S_i $S_i,S_j<A_k$	Shift S_i and S_j left A_k places

This instruction reads two 64 bit operands from S_i and S_j. The data is concatenated in a 128 bit field with the lowest order bit of S_i next to the highest order bit of S_j. The data is then shifted to the left, and a 64 bit field is returned to S_i. The result field is taken from the 64 bit window corresponding to the original S_i data.

The data is shifted left in an open-ended manner. That is, zero bits are inserted from the right as bits shift off to the left.

The shift count is read from A_k. The address register content is treated as a 32 bit positive integer. Shift counts greater than 128 result in a zero data field. A shift count of 64 results in the S_j data. A shift count of zero results in the original S_i data.

Issue Conditions:

- A_k free in phase 3
- S_j free in phase 5
- S_i free in phase 5
- Scalar register destination path free in phase 14

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 15

Instruction 113

Machine Code	CAL Instruction	Description
113ijk	S_i $S_j,S_i>A_k$	Shift S_i and S_j right A_k places

This instruction reads two 64 bit operands from S_i and S_j. The data is concatenated in a 128 bit field with the lowest order bit of S_j next to the highest order bit of S_i data. The data is then shifted to the right and a 64 bit field is returned to S_i. The result field is taken from the 64 bit window corresponding to the original S_i data.

The data is shifted right in an open-ended manner. That is, zero bits are inserted from the left as bits shift off to the right.

The shift count is read from A_k. The address register content is treated as a 32 bit positive integer. Shift counts greater than 128 result in a zero data field. A shift count of 64 results in the S_j data. A shift count of zero results in the original S_i data.

Issue Conditions:

- A_k free in phase 3
- S_j free in phase 5
- S_i free in phase 5
- Scalar register destination path free in phase 14

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 15

Instruction 114

Machine Code	CAL Instruction		Description
114i--	S_i	VM	Copy VM to S_i

This instruction reads the 64 bit mask from the VM register and enters it into S_i.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 14
- VM register in Vector logical unit free in phase 6
- Vector logical unit free in phase 6
- If an enter VM from S_j (034) instruction issues immediately prior to this instruction, issue will be held for four phases (two clock periods)
- Scalar register destination path free in phase 14

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 15

Instruction 115

Machine Code	CAL Instruction		Description
115i--	S_i	RT	Copy real time count to S_i

This instruction reads the 64 bit RTC and enters the value into S_i.

Issue Conditions:

- S_i free in phase 5
- RTC in Vector logical unit free in phase 6
- Vector logical unit free in phase 6
- If an enter RTC from S_j (035) instruction issues immediately prior to this instruction, issue will be held for four phases (two clock periods)
- Scalar register destination path free in phase 14

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 15

Instruction 116

Machine Code	CAL Instruction		Description
116ijk	S_i	jk,S,P	Load 6 bit positive value

This instruction forms a 64 bit word from the jk data in the instruction parcel. The low order six bits are copied from the instruction parcel. The higher order bits are zero. The result data is delivered to the S_i register.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 6

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 7

Instruction 117

Machine Code	CAL Instruction		Description
117ijk	S_i	jk,S,M	Load 6 bit negative value

This instruction forms a 64 bit word from the jk data in the instruction parcel. The low order six bits are copied from the instruction parcel. The higher order bits are ones. The result data is delivered to the S_i register.

Issue Conditions:

- S_i free in phase 5
- Scalar register destination path free in phase 6

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 7

Instruction 120

Machine Code	CAL Instruction		Description
120ijk	S_i	S_j+FS_k	Floating-point sum of S_j and S_k

This instruction forms the 64 bit floating-point sum of two 64 bit floating-point operands. The operands are read from S_j and S_k. The result is delivered to S_i. See description of floating-point add unit for details of special case treatment.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- floating-point add unit free in phase 8
- Scalar register destination path free in phase 28

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 29

Instruction 121

Machine Code	CAL Instruction		Description
121ijk	S_i	S_j-FS_k	Floating-point difference of S_j and S_k

This instruction forms the 64 bit floating-point difference of two 64 bit floating-point operands. The minuend is read from S_j and the subtrahend from S_k. The result is delivered to S_i. See description of the floating-point add unit for details of special case treatment.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- floating-point add unit free in phase 8
- Scalar register destination path free in phase 28

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 29

Instruction 122

Machine Code	CAL Instruction		Description
122i-k	S_i	FIX,S_k	Convert S_k to integer

This instruction reads a floating-point operand from S_k and delivers a fixed-point result to S_i. The conversion from floating point to fixed point is accomplished by adding the operand to a constant in the floating-point add unit. The result is then sign extended to form a 64 bit integer. See the description of floating-point add unit for details and special cases.

Issue Conditions:

- S_k free in phase 5
- S_i free in phase 5
- floating-point add unit free in phase 8
- Scalar register destination path free in phase 28

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 29

Instruction 123

Machine Code	CAL Instruction		Description
123i-k	S_i	FLT,S_k	Convert S_k to floating-point

This instruction reads a fixed-point operand from S_k and delivers a floating-point result to S_i. The conversion from fixed point to floating point is accomplished by adding the operand to a constant in the floating-point add unit. See the description of floating-point add unit for details and special cases.

Issue Conditions:

- S_k free in phase 5
- S_i free in phase 5
- floating-point add unit free in phase 8
- Scalar register destination path free in phase 28

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 29

Instructions 124 & 125

Machine Code	CAL Instruction		Description
124ijk	S_i	S_j*FS_k	Floating-point product of S_j and S_k
125ijk	S_i	S_j*FS_k	Same as instruction 124

This instruction forms the 64 bit floating-point product of two 64 bit floating-point operands. The operands are read from S_j and S_k. The result is delivered to S_i. See description of floating-point multiply unit for details of special case treatment.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- Floating-point multiply unit free in phase 8
- Scalar register destination path free in phase 30

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 31

Instruction 126

Machine Code	CAL Instruction	Description
126ijk	S_i S_j*IS$_k$	Reciprocal iteration 2-S_j*S_k

This instruction forms a floating-point quantity used in the reciprocal iteration algorithm. Operands are read from S_j and S_k. The result is delivered to S_i. See the description of floating-point multiply unit for the details of the reciprocal algorithm.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- Floating-point multiply unit free in phase 8
- Scalar register destination path free in phase 30

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 31

Instruction 127

Machine Code	CAL Instruction		Description
127ijk	S_i	S_j*QS$_k$	Square root iteration (3-S_j*S_k)/2

This instruction forms a floating-point quantity used in the reciprocal square root iteration algorithm. Operands are read from S_j and S_k. The result is delivered to S_i. See the description of floating-point multiply unit for the details of the square root algorithm.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- S_i free in phase 5
- Floating-point multiply unit free in phase 8
- Scalar register destination path free in phase 30

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 31

Instruction 130

Machine Code	CAL Instruction		Description
130ij-	S_i	A_j	Copy A_j to S_i

This instruction reads a 32 bit operand from A_j and enters it into S_i. The operand is extended to 64 bits by adding zeros in the upper half of the field.

Issue Conditions:

- A_j free in phase 3
- S_i free in phase 5
- Scalar register destination path free in phase 10

Completion Times:

- Next instruction in issue position in phase 2
- S_i free in phase 11

Instruction 131

Machine Code	CAL Instruction		Description
131ij-	S_i	$+A_j$	Copy A_j to S_i with sign extension

This instruction reads a 32 bit operand from A_j and enters it into S_i. The operand is extended to 64 bits by adding sign bits in the upper half of the field.

Issue Conditions:

- A_j free in phase 3
- S_i free in phase 5
- Scalar register destination path free in phase 10

Completion Times:

- Next instruction in issue position in one clock period
- S_i free in phase 11

Instruction 132

Machine Code	CAL Instruction	Description
132ij-	S_i /HS_j	Reciprocal approximation of S_j

This instruction forms a floating-point quantity which is a first approximation of the reciprocal of a floating-point operand. The operand is read from S_j. The result is delivered to S_i. See the description of floating-point multiply unit for details of the reciprocal algorithm.

Issue Conditions:

- S_j free in phase 5
- S_i free in phase 5
- Floating-point multiply unit free in phase 8
- Scalar register destination path free in phase 38

Completion Times:

- Next instruction in issue position in phase 2
- Floating-point multiply unit free in phase 10 for Scalar/Vector reciprocal instructions
- Floating-point multiply unit free in phase 18 for Scalar/Vector Floating-Point Multiply instructions
- S_i free in phase 39

Instruction 133

Machine Code	CAL Instruction		Description
133ij-	S_i	*QS_j	Square root approximation of S_j

This instruction forms a floating-point quantity which is a first approximation of the reciprocal square root of a floating-point operand. The operand is read from S_j. The result is delivered to S_i. See the description of floating-point multiply unit for details of the square root algorithm.

Issue Conditions:

- S_j free in phase 5
- S_i free in phase 5
- Floating-point multiply unit free in phase 8
- Scalar register destination path free in phase 38

Completion Times:

- Next instruction in issue position in phase 2
- Floating-point multiply unit free in phase 10 for Scalar/Vector reciprocal instructions
- Floating-point multiply unit free in phase 18 for Scalar /Vector Floating-Point Multiply instructions
- S_i free in phase 39

Instruction 134

Machine Code	CAL Instruction	Description
134ijk	V_i $<A_j,A_k>$	Two-port read V_i from CM (A_j) stride A_k

This instruction reads a vector stream of 64 bit words from CM into V_i. The length of the stream is determined by the content of the vector length register. This instruction issues immediately as a null function if the vector length is zero.

The first element address for the CM reference is formed by adding the content of A_j to the BP base address. The following addresses of the CM reference are separated by constant increments. The increment value is read from A_k.

The memory reference is made in a dual-port mode. In this mode the read and write functions of the CM port are allowed to proceed independent of one another.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- V_i free in phase 5
- No CM scalar (60-67) instruction issued in last 20 phases (10 clock periods)
- Memory port not active with a single-port CM vector operation (70-73)
- Memory read port not active with previous (134) instruction
- Memory port not active with fetch or foreground reference
- No memory port backup

Completion Times:

- Next instruction in issue position in phase 2
- Memory read port is free in 26 phases (13 clock periods)†. This completion time assumes no memory port backup occurs.
- V_i is free in phase 85†. This completion time assumes no delays in the memory bank access.

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

Instruction 135

Machine Code	CAL Instruction	Description
135ijk	<A_j,A_k> V_i	Two-port write V_i to CM (A_j) stride A_k

This instruction stores a vector stream of 64 bit words into CM from V_i. The length of the stream is determined by the content of the vector length register. This instruction issues immediately as a null function if the vector length is zero.

The first element address for the CM reference is formed by adding the content of A_j to the BP base address. The following addresses of the CM reference are separated by constant increments. The increment value is read from A_k.

The memory reference is made in a dual-port mode. In this mode the read and write functions of the CM port are allowed to proceed independent of one another.

Issue Conditions:

- A_j free in phase 3
- A_k free in phase 3
- V_i free in phase 5
- No CM scalar (60-67) instruction issued in last 20 phases (10 clock periods)
- Memory port not active with a single-port CM vector operation (70-73)
- Memory write port not active with previous (135) instruction
- Memory port not active with fetch or foreground reference
- No memory port backup.

Completion Times:

- Next instruction in issue position in phase 2
- Memory write port is free in 26 phases (13 clock periods)†. This completion time assumes no memory port backup occurs.
- V_i is free in phase 17††.

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instructions 136 & 137

Machine Code	CAL Instruction	Description
136ijk		Reserved for future use
137ijk		Reserved for future use

Currently execute as no-ops.

Completion Times:

- Next instruction in issue position in phase 2

Instruction 140

Machine Code	CAL Instruction		Description
140ijk	V_i	$S_j\&V_k$	Logical products of S_j and V_k

This instruction sequence reads a stream of vector elements from V_k, processes the data in the vector logical unit and delivers a stream of result elements to V_i. Data is read from S_j and is held in the vector logical unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. A bit-by-bit logical product is formed between the scalar word of data and each vector element. The resulting 64 logical products are then delivered as one element to the destination stream.

Issue Conditions:

- S_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector logical unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_k free for use as a source register in phase 17†
- V_i free for use as a source register in phase 21††
- Vector logical unit free in phase 22††

† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

†† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

Instruction 141

Machine Code	CAL Instruction		Description
141ijk	V_i	$V_j\&V_k$	Logical products of V_j and V_k

This instruction reads two streams of vector elements, processes them in the vector logical unit and delivers a stream of result elements to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. A bit-by-bit logical product is formed between the two arriving elements of the source vectors. The resulting 64 logical products are then delivered as one element to the destination stream.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector logical unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 21†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase 17††
- Vector logical unit free in phase 22††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 142

Machine Code	CAL Instruction		Description
142ijk	V_i	$S_j \backslash V_k$	Logical differences of S_j and V_k

This instruction sequence reads a stream of vector elements from V_k, processes the data in the vector logical unit and delivers a stream of result elements to V_i. Data is read from S_j and is held in the vector logical unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. A bit-by-bit logical difference is formed between the scalar data and each vector element. The resulting 64 logical differences are then delivered as one element to the destination stream.

Issue Conditions:

- S_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector logical unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 21†
- V_k free for use as a source register in phase 17††
- Vector logical unit free in phase 22††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 143

Machine Code	CAL Instruction		Description
143ijk	V_i	$V_j \backslash V_k$	Logical differences of V_j and V_k

This instruction reads two streams of vector elements, processes them in the vector logical unit and delivers a stream of result elements to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. A bit-by-bit logical difference is formed between the two arriving elements of the source vectors. The resulting 64 logical differences are delivered as one element to the destination stream.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector logical unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 21†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase 17††
- Vector logical unit free in phase 22††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 144

Machine Code	CAL Instruction		Description
144ijk	V_i	$S_j!V_k$	Logical sums of S_j and V_k

This instruction sequence reads a stream of vector elements from V_k, processes the data in the vector logical unit and delivers a stream of result elements to V_i. Data is read from S_j and is held in the vector logical unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. A bit-by-bit logical sum is formed between the scalar word of data and each vector element. The resulting 64 logical sums are then delivered as one element to the destination stream.

Issue Conditions:

- S_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector logical unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 21†
- V_k free for use as a source register in phase 17††
- Vector logical unit free in phase 22††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 145

Machine Code	CAL Instruction		Description
145ijk	V_i	$V_j!V_k$	Logical sums of V_j and V_k

This instruction reads two streams of vector elements, processes them in the vector logical unit and delivers a stream of result elements to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. A bit-by-bit logical sum is formed between the two arriving elements of the source vectors. The resulting 64 logical sums are delivered as one element to the destination stream.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector logical unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 21†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase 17††
- Vector logical unit free in phase 22††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 146

Machine Code	CAL Instruction		Description
146ijk	V_i	$S_j!V_k\&VM$	Merge S_j (VM=1) and $V_{k\ (VM=0)}$

This instruction sequence reads a stream of vector elements from V_k, processes the data in the vector logical unit and delivers a stream of result elements to V_i. Data is read from S_j and is held in the vector logical unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

The VM register is used as a control mechanism to select either the scalar register data or the vector element data as each element arrives at the vector logical unit. A bit of VM register data is associated with each element. The highest order bit of VM data is associated with the first vector element. The following bits of VM register data correspond with the following vector elements. The scalar register data is selected as a result element if the VM register contains a one in the designated element position. The V_k element is selected as a result element if the VM register contains a zero in the designated element position.

Issue Conditions:

- S_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector logical unit free in phase 5
- VM register in vector logical unit free in phase 6

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 21†
- V_k free for use as a source register in phase 17††
- Vector logical unit free in phase 22††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 147

Machine Code	CAL Instruction	Description
147ijk	V_i $V_j!V_k\&VM$	Merge V_j (VM=1) and $V_{k\ (VM=0)}$

This instruction reads two streams of vector elements, processes them in the vector logical unit and delivers a stream of result elements to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

The VM register is used as a control mechanism to select either the V_j data or the V_k data as each element pair arrive at the vector logical unit. A bit of VM register data is associated with each element. The highest order bit of VM data is associated with the first vector element. The following bits of VM data correspond with the following vector elements. The V_j data is selected as a result element if the VM register contains a one in the designated element position. The V_k data is selected as a result element if the VM register contains a zero in the designated element position.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector logical unit free in phase 5
- VM register in vector logical unit free in phase 5

Completion Times:

- Next instruction in issue position in phase 5
- V_i free for use as a source register in phase 21†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase 17††
- Vector logical unit free in phase 22††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 150

Machine Code	CAL Instruction		Description
150ijk	V_i	$V_j<A_k$	Shift V_j left A_k bits

This instruction sequence reads a stream of vector elements from V_j, processes the data in the vector integer unit and delivers a stream of result elements to V_i. Data is read from A_k and is held in the vector integer unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. Each element is shifted left by the number of bit positions indicated by the A_k value. Zero bits are inserted from the right as bits shift off to the left.

The A_k content is treated as a 32 bit positive integer. Shift counts greater than 64 result in a zero data field.

Issue Conditions:

- A_k free in phase 3
- V_j free in phase 5
- V_i free in phase 5
- Vector shift/pop/LZC unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_j free for use as a source register in phase 17††
- Vector shift/pop/LZC unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 151

Machine Code	CAL Instruction		Description
151ijk	V_i	$V_j>A_k$	Shift V_j right A_k bits

This instruction sequence reads a stream of vector elements from V_j, processes the data in the vector integer unit and delivers a stream of result elements to V_i. Data is read from A_k and is held in the vector integer unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. Each element is shifted right by the number of bit positions indicated by the A_k value. Zero bits are inserted from the left as bits shift off to the right.

The A_k content is treated as a 32 bit positive integer. Shift counts greater than 64 result in a zero data field.

Issue Conditions:

- A_k free in phase 3
- V_j free in phase 5
- V_i free in phase 5
- Vector shift/pop/LZC unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_j free for use as a source register in phase 17††
- Vector shift/pop/LZC unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 152

Machine Code	CAL Instruction		Description
152ijk	V_i	V_j, $V_j<A_k$	Continuous shift V_j left A_k places

This instruction sequence reads a stream of vector elements from V_j, processes the data in the vector integer unit and delivers a stream of result elements to V_i. Data is read from A_k and is held in the vector integer unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

The elements of data from V_j are processed in pairs for this sequence. Each element is concatenated with the following element, and the resulting 128 bit field is shifted left by the number of bit positions indicated in the A_k data. A 64 bit field from the original element window is then delivered to the destination vector stream. Specific examples of element steps are as follows:

The first element of V_j data is positioned in the higher portion of the 128 bit shift field. The second element of V_j data is positioned in the lower portion of the 128 bit shift field. The 128 bit field then shifts left by the amount of the shift count. A first result element is read from that portion of the 128 bit field originally occupied by the first element of data.

The second element of V_j data is then positioned in the higher portion of the 128 bit shift field. The third element of V_j data is entered in the lower portion of the field. This 128 bit field is then shifted left by the amount of the shift count. A second result element is read from the upper portion of the 128 bit field originally occupied by the second element of data.

This process continues until the last element of data is entered in the upper portion of the 128 bit shift field. A zero field is entered in the lower portion. This 128 bit field is then shifted left by the amount of the shift count. The last result element is read from the upper portion of the shift field.

The A_k content is treated as a 32 bit positive integer. Shift counts greater than 128 result in a zero data field. Zero bits are inserted at the right end of the 128 bit shift field as bits are shifted off to the left.

Issue Conditions:

- A_k free in phase 3
- V_j free in phase 5
- V_i free in phase 5
- Vector shift/pop/LZC unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_j free for use as a source register in phase 17††
- Vector shift/pop/LZC unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 153

Machine Code	CAL Instruction		Description
153ijk	V_i	$V_j, V_j>A_k$	Continuous shift V_j right A_k places

This instruction sequence reads a stream of vector elements from V_j, processes the data in the vector integer unit and delivers a stream of result elements to V_i. Data is read from A_k and is held in the vector integer unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

The elements of data from V_j are processed in pairs for this sequence. Each element is concatenated with the preceding element, and the resulting 128 bit field is shifted right by the number of bit positions indicated in the A_k data. A 64 bit field from the original element window is then delivered to the destination vector stream. Specific examples of element steps are as follows.

The first element of V_j data is positioned in the lower portion of the 128 bit shift field. The upper portion of the 128 bit shift field is cleared. The 128 bit field then shifts to the right by the amount of the shift count. A first result element is read from the lower portion of the 128 bit field originally occupied by the first element of data.

The second element of V_j data is then positioned in the lower portion of the 128 bit shift field. The first element of V_j data is entered in the upper portion of the field. This 128 bit field is then shifted right by the amount of the shift count. A second result element is read from the lower portion of the 128 bit field originally occupied by the second element of data.

This process continues until the last element of data is entered in the lower portion of the 128 bit shift field. The preceding element is entered in the upper portion. This 128 bit field is then shifted right by the amount of the shift count. The last result element is read from the lower portion of the field.

The A_k content is treated as a 32 bit positive integer. Shift counts greater than 128 result in a zero data field. Zero bits are inserted at the left end of the 128 bit shift field as bits are shifted off to the right.

Issue Conditions:

- A_k free in phase 3
- V_j free in phase 5
- V_i free in phase 5
- Vector shift/pop/LZC unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_j free for use as a source register in phase 17††
- Vector shift/pop/LZC unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 154

Machine Code	CAL Instruction		Description
154ijk	V_i	S_j*FV$_k$	Floating-point products of S_j and V_k

This instruction sequence reads a stream of vector elements from V_k, processes the data in the floating-point multiply unit and delivers a stream of result elements to V_i. Data is read from S_j and is held in the floating-point multiply unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. The floating-point multiply unit forms the 64 bit floating-point product of the arriving vector element and the scalar operand held in the unit. The result element is delivered to V_i. See description of floating-point multiply unit for details and special case treatment.

Issue Conditions:

- S_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Floating-point multiply unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 39†
- V_k free for use as a source register in phase 17††
- Floating-point multiply unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 155

Machine Code	CAL Instruction		Description
155ijk	V_i	V_j*FV$_k$	Floating-point products of V_j and V_k

This instruction reads two streams of vector elements, processes them in the floating-point multiply unit and delivers a result stream to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

The floating-point multiply unit forms a 64 bit floating-point product for each pair of arriving vector elements. The result element is delivered to V_i. See description of floating-point multiply unit for details and special case treatment.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Floating-point multiply unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 39†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase17††
- Floating-point multiply unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 156

Machine Code	CAL Instruction		Description
156ijk	V_i	V_j*IV_k	Reciprocal iteration 2-V_j*V_k

This instruction reads two streams of vector elements, processes them in the floating-point multiply unit and delivers a result stream to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

The floating-point multiply unit forms a 64 bit floating-point quantity used in the reciprocal iteration algorithm from each pair of arriving vector elements. The result element is delivered to V_i. See description of floating-point multiply unit for details and special case treatment.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Floating-point multiply unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 39†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase17††
- Floating-point multiply unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 157

Machine Code	CAL Instruction		Description
157ijk	V_i	V_j*QV_k	Square root iteration (3-V_j*$V_{k)/2}$

This instruction reads two streams of vector elements, processes them in the floating-point multiply unit and delivers a result stream to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

The floating-point multiply unit forms a 64 bit floating-point quantity used in the reciprocal square root iteration algorithm from each pair of arriving elements. The result element is delivered to V_i. See description of floating-point multiply unit for details and special case treatment.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Floating-point multiply unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 39†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase17††
- Floating-point multiply unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 160

Machine Code	CAL Instruction		Description
160ijk	V_i	S_j+V_k	Integer sums of S_j and V_k

This instruction sequence reads a stream of vector elements from V_k, processes the data in the vector integer unit and delivers a stream of result elements to V_i. Data is read from S_j and is held in the vector integer unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. The vector integer unit forms the integer sum of the scalar register data and each vector element. The result is delivered as one element of the destination stream.

Issue Conditions:

- S_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector integer unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_k free for use as a source register in phase 17††
- Vector integer unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 161

Machine Code	CAL Instruction		Description
161ijk	V_i	V_j+V_k	Integer sums of V_j and V_k

This instruction reads two streams of vector elements, processes them in the vector integer unit and delivers a stream of result elements to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. The vector integer unit forms the integer sum of the arriving elements of the source vectors. The result is delivered as one element of the destination stream.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector integer unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase 17††
- Vector integer unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 162

Machine Code	CAL Instruction		Description
162ijk	V_i	S_j-V_k	Integer differences of S_j and V_k

This instruction sequence reads a stream of vector elements from V_k, processes the data in the vector integer unit and delivers a stream of result elements to V_i. Data is read from S_j and is held in the vector integer unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. The vector integer unit forms the difference of the scalar register data minus each vector element. The result is delivered as one element of the destination stream.

Issue Conditions:

- S_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector integer unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_k free for use as a source register in phase 17††
- Vector integer unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 163

Machine Code	CAL Instruction		Description
163ijk	V_i	V_j-V_k	Integer differences of V_j and V_k

This instruction reads two streams of vector elements, processes them in the vector integer unit and delivers a stream of result elements to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. The vector integer unit forms the difference of the arriving minuend from V_j and the arriving subtrahend from V_k. The result is delivered as one element of the destination stream.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Vector integer unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase 17††
- Vector integer unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 164

Machine Code	CAL Instruction		Description
164ij0	V_i	PV_j	Population counts of V_j
164ij1	V_i	QV_j	Population count parities of V_j

This instruction sequence reads a stream of vector elements from V_j, processes the data in the vector integer unit and delivers a stream of result elements to V_i. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. The vector integer unit counts the number of one bits in each vector element. This count is delivered as a positive integer to the result stream.

If the lowest order bit of the k designator is set, the seven bit results from the population count are masked so that only the lowest order bit is delivered to the destination vector register. This serves as an odd parity bit for each vector register element.

Issue Conditions:

- V_j free in phase 5
- V_i free in phase 5
- Vector shift/pop/LZC unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_j free for use as a source register in phase 17††
- Vector shift/pop/LZC unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 165

Machine Code	CAL Instruction		Description
165ij-	V_i	ZV_j	Leading zero counts of V_j

This instruction sequence reads a stream of vector elements from V_j, processes the data in the vector integer unit and delivers a stream of result elements to V_i. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. The vector integer unit counts the number of leading zeros in each element. The element is considered as a field of 64 individual bits in this operation. This count is delivered as a positive integer to the result stream.

Issue Conditions:

- V_j free in phase 5
- V_i free in phase 5
- Vector shift/pop/LZC unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 27†
- V_j free for use as a source register in phase 17††
- Vector shift/pop/LZC unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 166

Machine Code	CAL Instruction		Description
166ij-	V_i	/HV_j	Reciprocal approximations of V_j

This instruction sequence reads a stream of vector elements from V_j, processes the data in the floating-point multiply unit and delivers a stream of result elements to V_i. The floating-point multiply unit forms a floating-point quantity, which is a first approximation to the reciprocal of the arriving vector element. The length of the vector stream is determined by the content of the vector length register. See the description of the floating-point multiply unit for details of the reciprocal algorithm.

Issue Conditions:

- V_j free in phase 5
- V_i free in phase 5
- Floating-point multiply unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 47†
- V_j free for use as a source register in phase 17††
- Floating-point multiply unit free in phase 28 for Scalar/Vector reciprocal instructions††
- Floating-point multiply unit free in phase 28 for Scalar/Vector Floating-Point Multiply instructions††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 167

Machine Code	CAL Instruction		Description
167ij-	V_i	*QV_j	Square root approximations of V_j

This instruction sequence reads a stream of vector elements from V_j, processes the data in the floating-point multiply unit and delivers a stream of result elements to V_i. The floating-point multiply unit forms a floating-point quantity, which is a first approximation to the reciprocal square root of the arriving vector element. The length of the vector stream is determined by the content of the vector length register. See the description of the floating-point multiply unit for details of the square root algorithm.

Issue Conditions:

- V_j free in phase 5
- V_i free in phase 5
- Floating-point multiply unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 47†
- V_j free for use as a source register in phase 17††
- Floating-point multiply unit free in phase 28 for Scalar/Vector reciprocal instructions††
- Floating-point multiply unit free in phase 28 for Scalar/Vector Floating-Point Multiply instructions††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 170

Machine Code	CAL Instruction		Description
170ijk	V_i	S_j+FV_k	Floating-point sums of S_j and V_k

This instruction sequence reads a stream of vector elements from V_k, processes the data in the floating-point add unit and delivers a stream of result elements to V_i. Data is read from S_j and is held in the floating-point add unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. The floating-point add unit forms the 64 bit floating-point sum of the arriving vector element and the scalar operand held in the unit. The result element is delivered to V_i. See description of floating-point add unit for details and special case treatment.

Issue Conditions:

- S_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Floating-point add unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 37†
- V_k free for use as a source register in phase 17††
- Floating-point add unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 171

Machine Code	CAL Instruction		Description
171ijk	V_i	V_j+FV_k	Floating-point sums of V_j and V_k

This instruction reads two streams of vector elements, processes them in the floating-point add unit and delivers a result stream to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

The floating-point add unit forms a 64 bit floating-point sum for each pair of arriving vector elements. The result element is delivered to V_i. See description of floating-point add unit for details and special case treatment.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Floating-point add unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 37†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase 17††
- Floating-point add unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 172

Machine Code	CAL Instruction		Description
172ijk	V_i	S_j-FV$_k$	Floating-point differences of S_j and V_k

This instruction sequence reads a stream of vector elements from V_k, processes the data in the floating-point add unit and delivers a stream of result elements to V_i. Data is read from S_j and is held in the floating-point add unit during the streaming operation. The length of the vector stream is determined by the content of the vector length register.

Each element of the vector stream is processed independent of the other elements in the stream. The floating-point add unit forms the 64 bit floating difference of the S_j minuend and the arriving vector element subtrahend. The result element is delivered to V_i. See description of floating-point add unit for details and special case treatment.

Issue Conditions:

- S_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Floating-point add unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 37†
- V_k free for use as a source register in phase 17††
- Floating-point add unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 173

Machine Code	CAL Instruction		Description
173ijk	V_i	V_j-FV_k	Floating-point differences of V_j and V_k

This instruction reads two streams of vector elements, processes them in the floating-point add unit and delivers a result stream to V_i. The source streams are from V_j and V_k. The length of each vector stream is determined by the content of the vector length register.

The floating-point add unit forms a 64 bit floating difference for each pair of arriving vector elements. The V_j data is the minuend, and the V_k data is the subtrahend. See description of floating-point add unit for details and special case treatment.

Issue Conditions:

- V_j free in phase 5
- V_k free in phase 5
- V_i free in phase 5
- Floating-point add unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 37†
- V_j free for use as a source register in phase 17††
- V_k free for use as a source register in phase 17††
- Floating-point add unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 174

Machine Code	CAL Instruction	Description
174i-k	V_i FIX, V_k	Convert V_k to integers

This instruction sequence reads a stream of vector elements from V_k, processes the data in the floating-point add unit and delivers a stream of result elements to V_i. The length of the vector stream is determined by the content of the vector length register.

The conversion from floating-point format to fixed-point format is accomplished by adding the operand to a constant in the floating-point add unit. The result is sign extended to form a 64 bit integer. See description of floating-point add unit for details and special case treatment.

Issue Conditions:

- V_k free in phase 5
- V_i free in phase 5
- Floating-point add unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 37†
- V_k free for use as a source register in phase 17††
- Floating-point add unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 175

Machine Code	CAL Instruction	Description
175i-k	V_i FLT, V_k	Convert V_k to floating-point

This instruction sequence reads a stream of vector elements from V_k, processes the data in the floating-point add unit and delivers a stream of result elements to V_i. The length of the vector stream is determined by the content of the vector length register.

The conversion from fixed-point format to floating-point format is accomplished by adding the operand to a constant in the floating-point add unit. The result is delivered to V_i. See description of floating-point add unit for details and special case treatment.

Issue Conditions:

- V_k free in phase 5
- V_i free in phase 5
- Floating-point add unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 37†
- V_k free for use as a source register in phase 17††
- Floating-point add unit free in phase 20††

† This time is for vector length = 1. Add two phases (one clock period) for each additional element beyond 1.

†† This time is for vector length = 1, 2, 3 or 4. Add two phases (one clock period) for each additional element beyond 4.

Instruction 176

Machine Code	CAL Instruction		Description
176ijk	V_i	CI, S_j&S_k	Compressed iota of mask S_j, stride S_k

This instruction forms a vector from two scalar operands. The first scalar operand is a 64 bit mask from S_j. The second scalar operand is a 64 bit stride from S_k. The operation is performed in two simultaneous stages: the generation of the iota vector, and the compression of the iota vector to form the result register.

The vector integer unit forms a 64 element iota vector from the stride. The iota vector's first element has a zero value, and subsequent elements are spaced by the stride increment. The sequence of element values in the iota vector is as follows:

0 x S_k, 1 x S_k, 2 x S_k, 3 x S_k, 4 x S_k, 5 x S_k, etc.

To form the iota vector the stride register (S_k) is captured and held in the vector integer unit. The initial value of the accumulated sum is cleared to zero, and the S_k value is then repeatedly added to the accumulated sum.

The 64 bit mask is used to compress elements of the iota vector as they are generated to from the result register. The mask register (S_j) is used to determine which elements of the iota vector will be delivered to the result register.

Starting with the highest order bit in the mask, an element of the iota vector is delivered to the result vector, where there is a one bit in the mask. An element of the iota vector is skipped and the position compressed, where there is a zero bit in the mask. Bits are then tested in order to the lowest order bit. The length of the resulting vector is then equal to the population count of one bits in the 64-bit mask (S_j). The remaining elements of the result register are unaffected.

When the remaining mask bits are zero, the instruction terminates. Execution time is variable depending on the mask content.

Issue Conditions:

- S_j free in phase 5
- S_k free in phase 5
- V_i free in phase 5
- Vector integer unit free in phase 8

Completion Times:

- Next instruction in issue position in phase 2
- V_i free for use as a source register in phase 29†
- Vector integer unit free in phase 30†

† This time is valid when the mask register (S_j) equals zero. Completion time is variable depending on the number of trailing zeros in the mask.

Instruction 177

Machine Code	CAL Instruction	Description
177---	PASS	No-op

Executes as a no-op.

Completion Times:

- Next instruction in issue position in one clock period

Chapter

4 Foreground Processor

A detailed description of the foreground processor and its instructions set is presented.

4.1 Foreground Processor Organization

The foreground processor controls background processor execution, generally by responding to background processor requests and controlling peripheral interfaces. It does not execute the system program code. The system program code resides in the common memory and is executed in background processors as required.

The foreground processor code is loaded at deadstart time from a file on the maintenance console. It cannot be altered during the operation of the system. This code is in the category generally referred to as firmware. Any error in program code is as fatal to system operation as hardware failure would be.

The primary function of the foreground processor program is to poll the four communication channel loops for requests from background processors and control communications on the four channel loops to facilitate data transfers between controllers on the channels such as mass storage transfers to memory. Many of the requests that will be recognized by the foreground processor require responses within a microsecond or less.

In order to achieve the desired response times in the foreground processor the program code is written in dedicated sections of in-line code. Each disk controller, for example, has its own section of code so that no computation is necessary to convert disk numbers to controller addresses. Each section of code is executed to the point where a channel function is issued. The sequence is then broken and the next program address for this section is sent to the channel controller called by the function and retained in its response address in the channel interface. The foreground processor looks for other work to do while the channel controller is executing the function.

The channel controller completing a function responds to a channel call by the foreground processor. The response includes the response address sent to the controller as part of the original function request. The next foreground processor instruction address is then sent from the channel controller to the foreground processor. The foreground processor jumps to the address and picks up the dropped sequence.

The foreground processor is divided into seven operational sections.

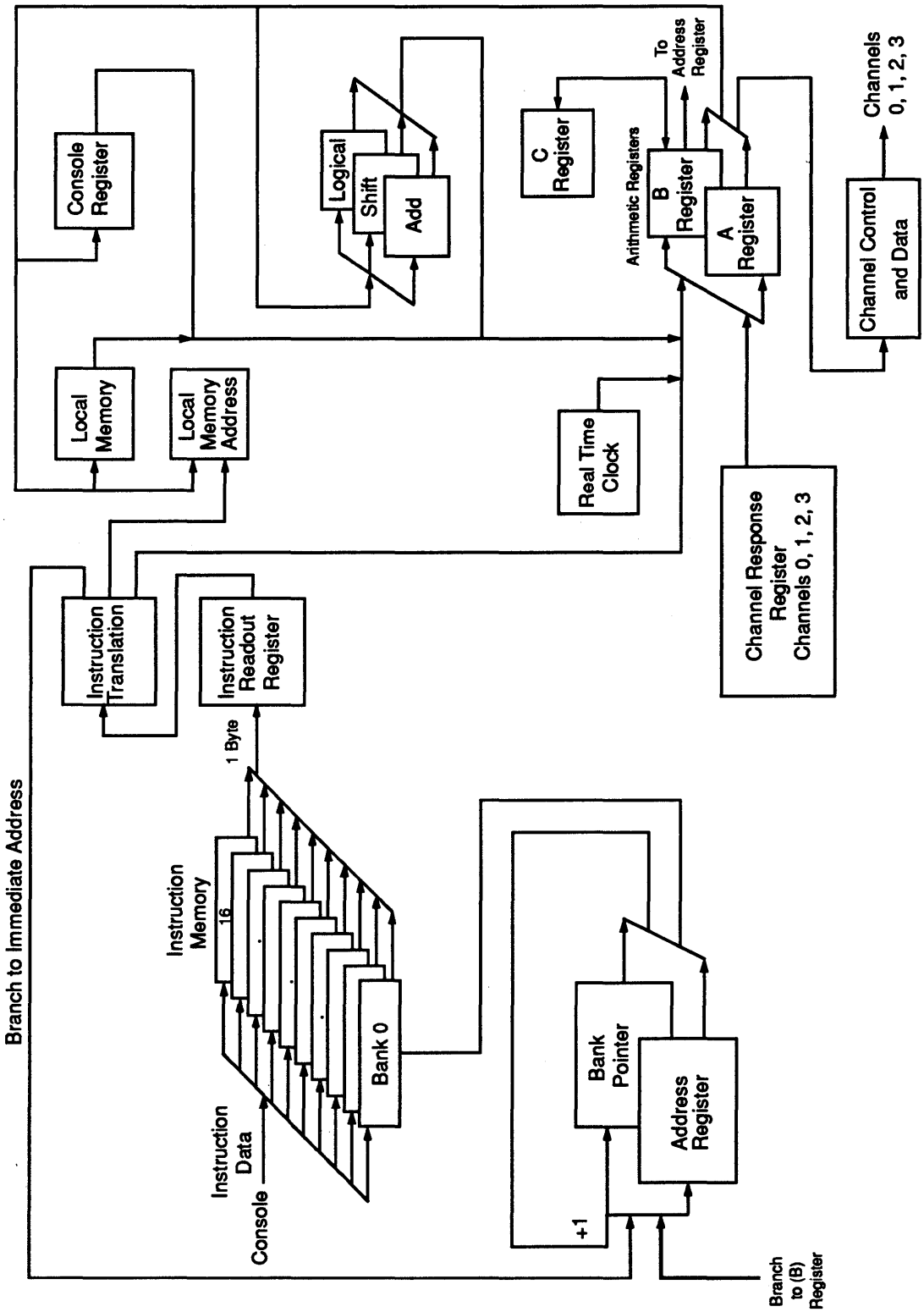
1. Deadstart circuits
2. Instruction issue mechanism
3. Local data memory
4. Channel communication
5. Functional units
6. Console communication
7. Real time clock

4.2 Deadstart Circuits

The foreground processor receives deadstart data from the maintenance console. The maintenance console is a microprocessor-based workstation. The link between them is an eight bit channel with full duplex paths. There are four control signals in each direction as well as the eight data signals. The signals are identified as follows:

Console to Foreground	Foreground to Console
1) Data ready	1) Data ready
1) Data resume	1) Data resume
1) Deadstart	1) Instruction parity error
1) Deadstart Mode	1) Local memory parity error
8) Data	8) Data

Figure 4 Foreground Processor Block Diagram



The deadstart process is controlled by a program executing in the maintenance console. The console transmits a deadstart signal to begin loading the foreground system after a power up situation. The deadstart signal remains on for some time as data is transmitted from the console to the foreground processor. Each byte of data is accompanied by a data ready signal. The data ready signal from the console remains active during the time the foreground processor resynchronizes the data and processes it. The foreground processor then sends a data resume signal to the console. The console program senses this signal and turns off the data ready signal. The foreground processor senses the data ready drop and turns off the data resume. This completes the exchange of one byte of data.

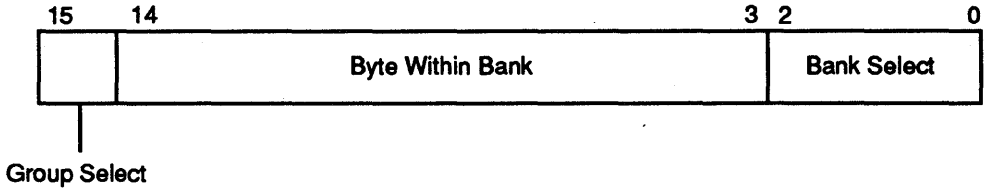
Each byte of data transmitted during the deadstart period may be directed by the console to one of two destinations. The mode selection line has a zero value to direct a byte to the foreground processor instruction memory. The mode selection line has a one value to direct a byte to the background processor reciprocal table memory. The foreground processor instruction memory is loaded beginning at address zero and continues until the deadstart signal is removed. All background processor table memories are loaded concurrently when data is directed to them.

The background processors and common memory are forced into an idle mode during the loading of the foreground processor memory. Foreground processor execution then begins at address zero in the foreground memory. The preamble of this program code clears common memory of noise data and presets nominal conditions in the background processors.

4.3 Instruction Issue

The foreground processor instruction memory contains 65,536 bytes of data. The instruction memory is divided into two groups of 32,768 bytes each. Each group consists of eight banks of 4096 bytes each. The lowest order three bits of the instruction memory address select the bank and the upper bit selects the group. The memory can read a new byte in sequence each clock period and advance the byte address.

Instruction Memory Address Scheme:.



Instructions issue directly from the instruction memory readout register. An instruction format may include one byte, two bytes, three bytes, or five bytes. Maximum instruction issue rate is one instruction every clock period (two clock phases). Multi-byte instructions read subsequent bytes every clock period.

Instruction sequences may branch to an address which is read from the instruction stream as a constant. Such instructions have a three byte format with the branch criterion in the first byte and the address in the following two bytes. Alternatively, the instruction sequence may branch to a computed address contained in the B register.

Constants may be entered into the computation in four bit, twelve bit, two byte, or four byte format. All arithmetic functions are performed in 32 bit integer form.

When integer values are read from the instruction stream and involve more than one byte the highest order data byte appears first.

4.4 Local Data Memory

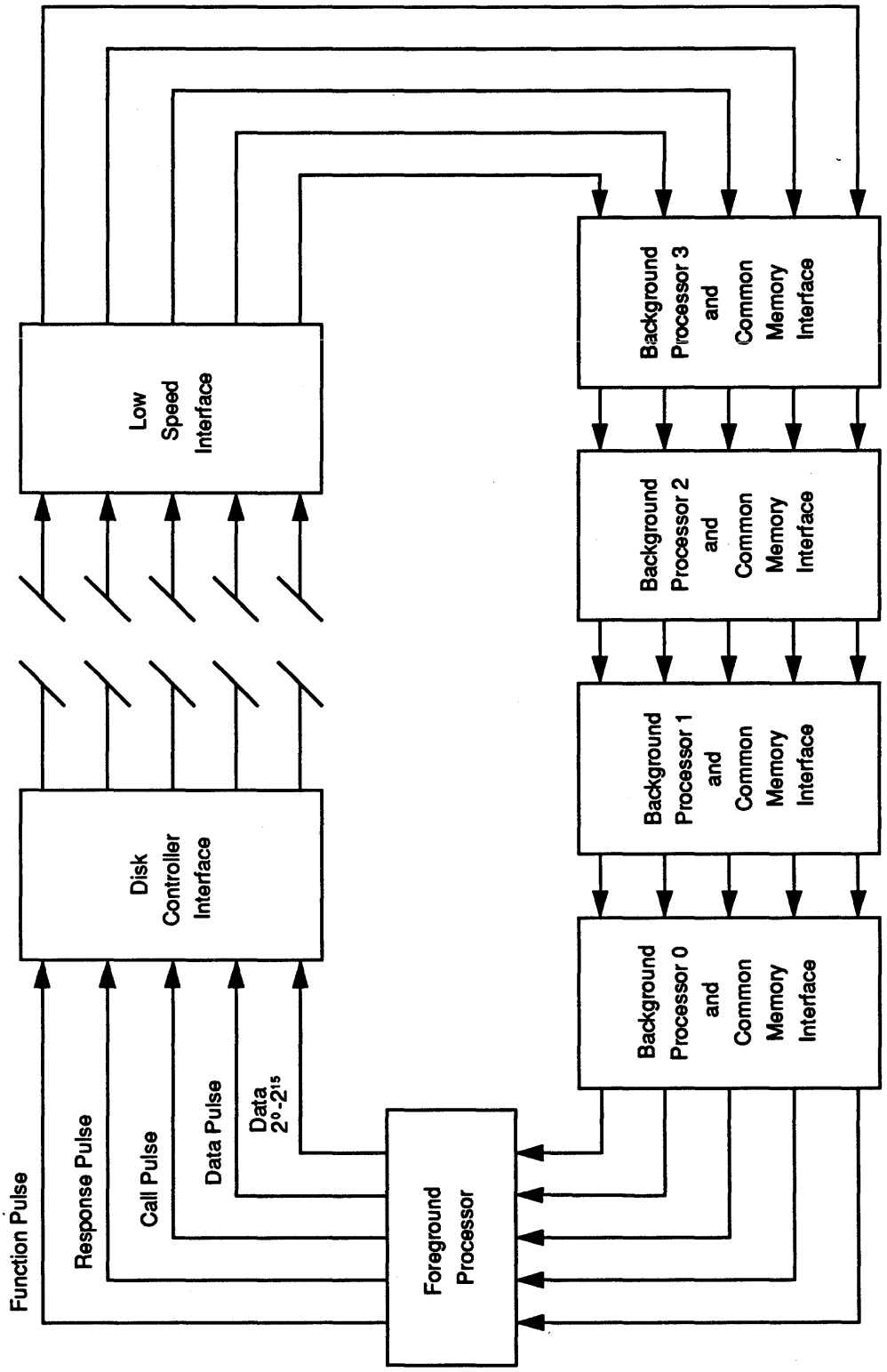
The foreground processor local memory contains 4096 words of data arranged in a single bank of 32 bit length. Data may be read from, or written into, this memory every four clock periods.

4.5 Channel Communication

The foreground processor interfaces with all four of the foreground system channels (see Figure 5 on page 186). Foreground processor instructions initiate channel activity. The foreground channel consists of 16 data lines and four control lines. One of these control lines, the data pulse line, is used exclusively between nodes on the channel. The foreground processor makes no use of this pulse and functions during this period as a simple way station for the 16 data lines.

The foreground processor has a channel busy flag associated with each of the four channels. This flag is used in program branch tests to synchronize the channel activity. A channel call pulse is sent from the foreground processor over the channel when the appropriate instruction is executed. The channel busy flag is set at this time. The channel call pulse passes each of the channel nodes before returning to the foreground processor. As the pulse passes each node the associated controller may gate interrupt data onto the data lines.

Figure 5 Communication Channel Loop



Controllers located further along the channel may replace the data with their own data. When the channel call pulse arrives at the foreground processor the busy flag is cleared and the data is entered into a 32 bit channel data register (only the low-order 16 bits may be non-zero). The foreground processor may then read the data and process it.

The foreground processor sends a function pulse over the channel to initiate a specific controller activity. This pulse is followed by two 32 bit words of data. The first word is called the function word. It contains the controller and function identification and supporting parameters. The second word is called the data word and contains computed data values from the foreground processor. Each controller on the channel tests the function code as the function pulse passes its node. A recognized code causes activity in that controller. The function pulse is terminated as it arrives back at the foreground processor.

The channel busy flag is set as a function pulse leaves the foreground processor for transmission over the channel. This flag is not cleared as the function pulse arrives at the foreground processor from the channel circuit. The addressed controller is responsible for sending a response pulse and a data word at the proper time to the foreground processor. This response pulse clears the channel busy flag and enters the channel data register with 32 bits of response data.

4.6 Functional Units

The foreground processor contains the following functional unit operations.

1. Add
2. Subtract
3. Shift left open ended
4. Shift right open ended
5. Logical product
6. Logical difference
7. Logical sum
8. Swap A & B
9. Copy C to B
10. Copy B to C

The operations are performed with the aid of two 32 bit registers. These registers are designated register A and register B. Except for the Copy B to C

operation, results are always left in the B register. The C register is a temporary 32 bit storage register accessible by the copy instructions.

The Add, Subtract and Shift functions require three clock periods (six phases) for completion. The Logical, Swap and Copy functions require one clock period (two phases) for completion.

4.7 Console Communication

The maintenance console is used to monitor system operation as well as to provide the deadstart data previously described. All functions require an operating foreground processor program.

The foreground processor has a 32 bit console data register which the foreground processor program may load with data from the A or B register. The execution of such an instruction sends a data ready pulse to the console along with the uppermost byte of data in the console data register. The console program can sense this signal and proceed with the exchange of control signals as described in the deadstart procedure.

The control signal exchange for the first byte of data ends with the release of the data resume signal from console to foreground processor. This event causes the console data register to advance a pointer and send another data ready signal and the second byte of data in the console data register. This process is repeated four times to transmit the 32 bit data word to the console.

Following the above exchange the console can begin the reverse exchange to transmit a 32 bit quantity from the console memory to the foreground processor. A full flag in the foreground processor indicates when the console data register is full of data. The foreground processor program can then read the data into the A or B register.

4.8 Real Time Clock

The foreground processor contains a 32 bit real time clock which advances one count every clock period. This clock is set to a zero value at the end of the deadstart period and counts continuously from that time onward. The value in the real time clock can be read into the A or B register. The background processors have similar 64 bit real time clock registers and all clocks are started with a zero value at the same clock period.

4.9 Error Checking

The foreground processor instruction memory is loaded with data from the console at deadstart time. Each byte of data written into the instruction memory is accompanied by a parity bit. This data is verified during normal foreground processor operation as each instruction byte is read from the memory. A parity error sets a flag in the foreground processor and an error signal is sent to the maintenance console.

The foreground processor local memory has a parity bit for each 32 bit data word in the memory. This parity bit is written along with the data on each write reference. The data is verified on each read reference. A parity error sets a flag in the foreground processor and an error signal is sent to the maintenance console.

Each byte of data transmitted from the maintenance console to the foreground processor is resynchronized at the foreground processor. When the foreground processor instruction memory is being written (deadstart true and deadstart mode false), each byte written is read from the memory and echoed back over the eight bit channel. In all other cases (deadstart false or deadstart and deadstart mode both true), each byte is echoed back over the channel from the synchronization register.

4.10 Foreground Processor Instruction Descriptions

FOREGROUND INSTRUCTION SET

Machine Code	Instruction Mnemonics		Description
000	J	label	Branch always
001	JB		Branch always to B register
002	JCOD	label	Branch if output to console done
003	JCIR	label	Branch if console input ready
004	JZ	label	Branch if B register equal zero
005	JN	label	Branch if B register not equal zero
006	JP	label	Branch if B register greater than zero
007	JM	label	Branch if B register less than zero
010	JCB	0, label	Branch if channel zero busy

Machine Code	Instruction Mnemonics		Description
011	JCB	1, label	Branch if channel one busy
012	JCB	2, label	Branch if channel two busy
013	JCB	3, label	Branch if channel three busy
014	JCI	0, label	Branch if channel zero idle
015	JCI	1, label	Branch if channel one idle
016	JCI	2, label	Branch if channel two idle
017	JCI	3, label	Branch if channel three idle
020	LUA	constant	Enter A with upper 16 bit constant
021	LLA	constant	Enter A with lower 16 bit constant
022	LA32	constant	Enter A with 32 bit constant
023	LA32	constant	Enter A with 32 bit constant
024	LAC	0	Enter A with channel zero data
025	LAC	1	Enter A with channel one data
026	LAC	2	Enter A with channel two data
027	LAC	3	Enter A with channel three data
030	LUB	constant	Enter B with upper 16 bit constant
031	LLB	constant	Enter B with lower 16 bit constant
032	LB32	constant	Enter B with 32 bit constant
033	LB32	constant	Enter B with 32 bit constant
034	LBC	0	Enter B with channel zero data
035	LBC	1	Enter B with channel one data
036	LBC	2	Enter B with channel two data
037	LBC	3	Enter B with channel three data
040	ADD		Enter B with integer sum A + B
041	SUB		Enter B with integer difference B - A
042	C	B	Enter C with B
043	B	C	Enter B with C
044	AND		Enter B with logical product A and B
045	XOR		Enter B with logical difference A and B
046	IOR		Enter B with logical sum A and B

Machine Code	Instruction Mnemonics		Description
047	SWP		Swap B and A
050	TF	0	Transmit function A,B on channel zero
051	TF	1	Transmit function A,B on channel one
052	TF	2	Transmit function A,B on channel two
053	TF	3	Transmit function A,B on channel three
054	GC	0	Transmit general call on channel zero
055	GC	1	Transmit general call on channel one
056	GC	2	Transmit general call on channel two
057	GC	3	Transmit general call on channel three
060	LACN		Enter A from console register
061	SACN		Store A to console register
062	LACL		Enter A with real time clock
063			Pass
064	LAA		Enter A from local memory address A
065	SAA		Store A into local memory address A
066	LAB		Enter A from local memory address B
067	SAB		Store A into local memory address B
070	LBCN		Enter B from console register
071	SBCN		Store B to console register
072	LBCL		Enter B with real time clock
073			Pass
074	LBA		Enter B from local memory address A
075	SBA		Store B into local memory address A
076	LBB		Enter B from local memory address B
077	SBB		Store B into local memory address B
100-137	SR	cnt	Enter B with B shifted right
140-177	SL	cnt	Enter B with B shifted left
200-217	LA4	constant	Enter A with 4 bit constant
220-237	LA12	constant	Enter A with 12 bit constant
240-257	LAM	label	Enter A from constant local memory address

Machine Code	Instruction Mnemonics		Description
260-277	SAM	label	Store A into constant local memory address
300-317	LB4	constant	Enter B with 4 bit constant
320-337	LB12	constant	Enter B with 12 bit constant
340-357	LBM	label	Enter B from constant local memory address
360-377	SBM	label	Store B into constant local memory address

Instruction 000

Machine Code	Instruction Mnemonics	Description
000	J label	Branch to constant address

This instruction causes the foreground processor program to branch unconditionally to a constant byte address. The constant address is contained in the two bytes from the instruction stream.

Issue Conditions:

- None

Completion Times:

- Next instruction in issue position in 12 clock periods

Instruction 001

Machine Code	Instruction Mnemonics	Description
001	JB	Branch to B register

This instruction causes the foreground processor program to branch unconditionally to a computed address. The computed address is obtained from the lowest order 16 bits of the B register.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 12 clock periods

Instruction 002

Machine Code	Instruction Mnemonics	Description
002	JCOD label	Branch to constant address if output to console done

This instruction causes the foreground processor program to branch to a constant address if the transfer of the console register to the console is complete. The constant address is contained in the two bytes following the instruction byte. The current program sequence is continued if the transfer is not complete.

Issue Conditions:

- None

Completion Times:

- Next instruction in issue position in 12 clock periods (branch taken)
- Next instruction in issue position in 3 clock periods (fall-through)

Instruction 003

Machine Code	Instruction Mnemonics	Description
003	JCIR label	Branch to constant address if console input ready

This instruction causes the foreground processor program to branch to a constant address if the console input data register is filled. The constant address is contained in the two bytes following the instruction byte. The current program sequence is continued if the console register is not filled.

Issue Conditions:

- None

Completion Times:

- Next instruction in issue position in 12 clock periods (branch taken)
- Next instruction in issue position in 3 clock periods (fall-through)

Instruction 004

Machine Code	Instruction Mnemonics		Description
004	JZ	label	Branch to constant address if B equals zero

This instruction causes the foreground processor program to branch to a constant address if the number in the B register is zero. The constant address is contained in the two bytes following the instruction byte. The current program sequence is continued if the B register content is nonzero.

Issue Conditions:

- B register free and not used as a destination register in the previous 2 clock periods

Completion Times:

- Next instruction in issue position in 12 clock periods (branch taken)
- Next instruction in issue position in 3 clock periods (fall-through)

Instruction 005

Machine Code	Instruction Mnemonics		Description
005	JN	label	Branch to constant address if B is nonzero

This instruction causes the foreground processor program to branch to a constant address if the number in the B register is nonzero. The constant address is contained in the two bytes following the instruction byte. The current program sequence is continued if the B register content is zero.

Issue Conditions:

- B register free and not used as a destination register in the previous 2 clock periods

Completion Times:

- Next instruction in issue position in 12 clock periods (branch taken)
- Next instruction in issue position in 3 clock periods (fall-through)

Instruction 006

Machine Code	Instruction Mnemonics		Description
006	JP	label	Branch to constant address if B is positive

This instruction causes the foreground processor program to branch to a constant address if the number in the B register is positive (0 is a positive integer in the foreground two's complement arithmetic). The constant address is contained in the two bytes following the instruction byte. The current program sequence is continued if the B register content is negative.

Issue Conditions:

- B register free and not used as a destination register in the previous 2 clock periods

Completion Times:

- Next instruction in issue position in 12 clock periods (branch taken)
- Next instruction in issue position in 3 clock periods (fall-through)

Instruction 007

Machine Code	Instruction Mnemonics	Description
007	JM label	Branch to constant address if B is negative

This instruction causes the foreground processor program to branch to a constant address if the number in the B register is negative. The constant address is contained in the two bytes following the instruction byte. The current program sequence is continued if the B register content is positive or zero.

Issue Conditions:

- B register free and not used as a destination register in the previous 2 clock periods

Completion Times:

- Next instruction in issue position in 12 clock periods (branch taken)
- Next instruction in issue position in 3 clock periods (fall-through)

Instructions 010 - 013

Machine Code	Instruction Mnemonics		Description
010	JCB	0, label	Branch to constant address if channel zero busy
011	JCB	1, label	Branch to constant address if channel one busy
012	JCB	2, label	Branch to constant address if channel two busy
013	JCB	3, label	Branch to constant address if channel three busy

These instructions cause the foreground processor program to branch to a constant address if the channel zero through three busy flags are set. The constant address is contained in the two bytes following the instruction byte. The current program sequence is continued if the channel zero through three busy flags are cleared.

Issue Conditions:

- None

Completion Times:

- Next instruction in issue position in 12 clock periods (branch taken)
- Next instruction in issue position in 3 clock periods (fall-through)

Instructions 014 - 017

Machine Code	Instruction Mnemonics		Description
014	JCI	0, label	Branch to constant address if channel zero idle
015	JCI	1, label	Branch to constant address if channel one idle
016	JCI	2, label	Branch to constant address if channel two idle
017	JCI	3, label	Branch to constant address if channel three idle

These instructions cause the foreground processor program to branch to a constant address if the channel zero through three busy flags are not set. The constant address is contained in the two bytes following the instruction byte. The current program sequence is continued if the channel zero through three busy flags are set.

Issue Conditions:

- None

Completion Times:

- Next instruction in issue position in 12 clock periods (branch taken)
- Next instruction in issue position in 3 clock periods (fall-through)

Instruction 020

Machine Code	Instruction Mnemonics		Description
020	LUA	constant	Enter upper A with 16 bit constant

This instruction reads a 16 bit constant from the next two bytes in the instruction stream and enters this value into the upper half of the A register. The lower half of the A register is cleared.

Issue Conditions:

- A register free

Completion Times:

- Next instruction in issue position in 3 clock periods
- A register is free in 3 clock periods

Instruction 021

Machine Code	Instruction Mnemonics		Description
021	LLA	constant	Enter lower A with 16 bit constant

This instruction reads a 16 bit constant from the next two bytes in the instruction stream and enters this value into the lower half of the A register. The upper half of the A register is cleared.

Issue Conditions:

- A register free

Completion Times:

- Next instruction in issue position in 3 clock periods
- A register is free in 3 clock periods

Instructions 022, 023

Machine Code	Instruction Mnemonics		Description
022, 023	LA32	constant	Enter A with 32 bit constant

These instructions read a 32 bit constant from the next four bytes in the instruction stream and enter this value into the A register.

Issue Conditions:

- A register free

Completion Times:

- Next instruction in issue position in 5 clock periods
- A register is free in 3 clock periods. An additional 2 clock periods are require to clear the data parcels from the instruction stream.

Instructions 024 - 027

Machine Code	Instruction Mnemonics		Description
024	LAC	0	Enter A with channel zero data
025	LAC	1	Enter A with channel one data
026	LAC	2	Enter A with channel two data
027	LAC	3	Enter A with channel three data

These instructions read 32 bits of data from the designated channel data register and enter that data into the A register. No test is made of the channel busy flag.

Issue Conditions:

- A register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 3 clock periods

Instruction 030

Machine Code	Instruction Mnemonics		Description
030	LUB	constant	Enter upper B with 16 bit constant

This instruction reads a 16 bit constant from the next two bytes in the instruction stream and enters this value into the upper half of the B register. The lower half of the B register is cleared.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 3 clock periods
- B register is free in 3 clock periods

Instruction 031

Machine Code	Instruction Mnemonics	Description
031	LLB constant	Enter lower B with 16 bit constant

This instruction reads a 16 bit constant from the next two bytes in the instruction stream and enters this value into the lower half of the B register. The upper half of the B register is cleared.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 3 clock periods
- B register is free in 3 clock periods

Instructions 032, 033

Machine Code	Instruction Mnemonics		Description
032, 033	LB32	constant	Enter B with 32 bit constant

These instructions read a 32 bit constant from the next four bytes in the instruction stream and enter this value into the B register.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 5 clock periods
- B register is free in 3 clock periods. An additional 2 clock periods are require to clear the data parcels from the instruction stream.

Instructions 034 - 037

Machine Code	Instruction Mnemonics		Description
034	LBC	0	Enter B with channel zero data
035	LBC	1	Enter B with channel one data
036	LBC	2	Enter B with channel two data
037	LBC	3	Enter B with channel three data

These instructions read 32 bits of data from the designated channel data register and enter that data into the B register. No test is made of the channel busy flag.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 3 clock periods

Instruction 040

Machine Code	Instruction Mnemonics	Description
040	ADD	Enter B with integer sum B + A

This instruction performs an integer add of the A register and the B register. The result is stored in the B register.

Issue Conditions:

- A register free
- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 4 clock periods

Instruction 041

Machine Code	Instruction Mnemonics	Description
041	SUB	Enter B with integer difference B - A

This instruction performs the integer subtraction of the A register from the B register. The result is stored in the B register.

Issue Conditions:

- A register free
- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 4 clock periods

Instruction 042

Machine Code	Instruction Mnemonics	Description
042	C B	Enter C with B

This instruction copies the contents of the B register into the C register. The C register is a temporary 32 bit storage register accessible by the 042 and 043 instructions.

Issue Conditions:

- B register free
- C register free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 1 clock period
- C register is free in 1 clock period

Instruction 043

Machine Code	Instruction Mnemonics	Description
043	B C	Enter B with C

This instruction copies the contents of the C register into the B register. The C register is a temporary 32 bit storage register accessible by the 042 and 043 instructions.

Issue Conditions:

- B register free
- C register free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 1 clock period
- C register is free in 1 clock period

Instruction 044

Machine Code	Instruction Mnemonics	Description
044	AND	Enter B with logical product A and B

This instruction forms the bit by bit logical product of the data in the A register and the data in the B register. The result is stored in the B register.

Issue Conditions:

- A register free
- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 1 clock period

Instruction 045

Machine Code	Instruction Mnemonics	Description
045	XOR	Enter B with logical difference A and B

This instruction forms the bit by bit logical difference of the data in the A register and the data in the B register. The result is stored in the B register.

Issue Conditions:

- A register free
- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 1 clock period

Instruction 046

Machine Code	Instruction Mnemonics	Description
046	IOR	Enter B with logical sum A and B

This instruction forms the bit by bit logical sum of the data in the A register and the data in the B register. The result is stored in the B register.

Issue Conditions:

- A register free
- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 1 clock period

Instruction 047

Machine Code	Instruction Mnemonics	Description
047	SWP	Swap A and B

This instruction swaps the data in the A register with the data in the B register. The exchanged data is in the respective registers.

Issue Conditions:

- A register free
- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 1 clock period

Instructions 050 - 053

Machine Code	Instruction Mnemonics		Description
050	TF	0	Transmit function A,B on channel zero
051	TF	1	Transmit function A,B on channel one
052	TF	2	Transmit function A,B on channel two
053	TF	3	Transmit function A,B on channel three

These instructions send a function pulse over the designated channel and follow the pulse with two 32 bit words of data. The first word (the function word) is read from register A, and the second word is read from register B. The channel busy flag is set and remains set until a channel node responds with a data word. This response word is entered into the designated channel data register and the channel busy flag is cleared.

Issue Conditions:

- A register free
- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 1 clock period

Instructions 054 - 057

Machine Code	Instruction Mnemonics		Description
054	GC	0	Transmit general call on channel zero
055	GC	1	Transmit general call on channel one
056	GC	2	Transmit general call on channel two
057	GC	3	Transmit general call on channel three

These instructions send a channel call pulse over the designated channel and follow the pulse with a blank 32 bit word of data. The channel busy flag is set and remains set until the channel call pulse has traversed the channel loop and returned to the foreground processor. The channel busy flag is then cleared and the following 32 bit word of data is entered into the channel data register.

Any channel node responding to the channel call pulse will replace the blank data word with interrupt response data previously set by the foreground processor program (via a transmit function).

Issue Conditions:

- None

Completion Times:

- Next instruction in issue position in 1 clock period

Instruction 060

Machine Code	Instruction Mnemonics	Description
060	LACN	Enter A from console register

This instruction reads a 32 bit data word from the console data assembly register and enters this data into the A register. The console assembly register full flag is also cleared. The input buffer full flag can be tested using a 003 instruction.

Issue Conditions:

- A register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 3 clock periods

Instruction 061

Machine Code	Instruction Mnemonics	Description
061	SACN	Store A into console register

This instruction reads a 32 bit data word from register A and enters this data into the console data disassembly register. The console register buffer full flag is set in this process and data transmission is initiated to the maintenance console. Upon completion of the transmission to the console, the buffer full flag is cleared telling the foreground processor that the transfer was completed. The output buffer full flag can be tested using a 002 instruction.

Issue Conditions:

- A register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period

Instruction 062

Machine Code	Instruction Mnemonics	Description
062	LACL	Enter A with real time clock

This instruction reads the current 32 bit count from the real time clock register and enters this value into the A register.

Issue Conditions:

- A register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 3 clock periods

Instruction 063

Machine Code	Instruction Mnemonics	Description
063		Pass

This instruction issues with no resulting computational activity.

Issue Conditions:

- None

Completion Times:

- Next instruction in issue position in 1 clock period

Instruction 064

Machine Code	Instruction Mnemonics	Description
064	LAA	Enter A from local memory address A

This instruction reads a 32 bit word from a local memory address and enters this data into the A register. The local memory address is obtained from the lowest order 12 bits of the A register.

Issue Conditions:

- A register free
- Local memory free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 6 clock periods
- Local memory is free in 4 clock periods

Instruction 065

Machine Code	Instruction Mnemonics	Description
065	SAA	Store A into local memory address A

This instruction reads a 32 bit word from the A register and enters this data into local memory. The local memory address is obtained from the lowest order 12 bits of the A register.

Issue Conditions:

- A register free
- Local memory free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- Local memory is free in 4 clock periods

Instruction 066

Machine Code	Instruction Mnemonics	Description
066	LAB	Enter A from local memory address B

This instruction reads a 32 bit word from a local memory address and enters this data into the A register. The local memory address is obtained from the lowest order 12 bits of the B register.

Issue Conditions:

- A register free
- B register free
- Local memory free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 6 clock periods
- B register is free in 1 clock period
- Local memory is free in 4 clock periods

Instruction 067

Machine Code	Instruction Mnemonics	Description
067	SAB	Store A into local memory address B

This instruction reads a 32 bit word from the A register and enters this data into local memory. The local memory address is obtained from the lowest order 12 bits of the B register.

Issue Conditions:

- A register free
- B register free
- Local memory free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 1 clock period
- Local memory is free in 4 clock periods

Instruction 070

Machine Code	Instruction Mnemonics	Description
070	LBCN	Enter B from console register

This instruction reads a 32 bit data word from the console data assembly register and enters this data into the B register. The console assembly register full flag is also cleared. The input buffer full flag can be tested using a 003 instruction.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 3 clock periods

Instruction 071

Machine Code	Instruction Mnemonics	Description
071	SBCN	Store B into console register

This instruction reads a 32 bit data word from register B and enters this data into the console data disassembly register. The console register buffer full flag is set in this process and data transmission is initiated to the maintenance console. Upon completion of the transmission to the console, the buffer full flag is cleared telling the foreground processor that the transfer was completed. The output buffer full flag can be tested using a 002 instruction.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 1 clock period

Instruction 072

Machine Code	Instruction Mnemonics	Description
072	LBCL	Enter B with real time clock

This instruction reads the current 32 bit count from the real time clock register and enters this value into the B register.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 3 clock periods

Instruction 073

Machine Code	Instruction Mnemonics	Description
073		Pass

This instruction issues with no resulting computational activity.

Issue Conditions:

- None

Completion Times:

- Next instruction in issue position in 1 clock period

Instruction 074

Machine Code	Instruction Mnemonics	Description
074	LBA	Enter B from local memory address A

This instruction reads a 32 bit word from a local memory address and enters this data into the B register. The local memory address is obtained from the lowest order 12 bits of the A register.

Issue Conditions:

- A register free
- B register free
- Local memory free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 6 clock periods
- Local memory is free in 4 clock periods

Instruction 075

Machine Code	Instruction Mnemonics	Description
075	SBA	Store B into local memory address A

This instruction reads a 32 bit word from the B register and enters this data into local memory. The local memory address is obtained from the lowest order 12 bits of the A register.

Issue Conditions:

- A register free
- B register free
- Local memory free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 1 clock period
- B register is free in 1 clock period
- Local memory is free in 4 clock periods

Instruction 076

Machine Code	Instruction Mnemonics	Description
076	LBB	Enter B from local memory address B

This instruction reads a 32 bit word from a local memory address and enters this data into the B register. The local memory address is obtained from the lowest order 12 bits of the B register.

Issue Conditions:

- B register free
- Local memory free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 6 clock periods
- Local memory is free in 4 clock periods

Instruction 077

Machine Code	Instruction Mnemonics	Description
077	SBB	Store B into local memory address B

This instruction reads a 32 bit word from the B register and enters this data into local memory. The local memory address is obtained from the lowest order 12 bits of the B register.

Issue Conditions:

- B register free
- Local memory free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 1 clock period
- Local memory is free in 4 clock periods

Instruction 100-137

Machine Code	Instruction Mnemonics		Description
100-137	SR	cnt	Enter B with B shifted right cnt places

This instruction reads the 32 bit data field from the B register, shifts the data open ended to the right, and stores the result back into the B register. The shift count is obtained from the lowest order five bits of the instruction byte. Zeros are filled from the left.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 4 clock periods

Instruction 140-177

Machine Code	Instruction Mnemonics	Description
140-177	SL cnt	Enter B with B shifted left cnt places

This instruction reads the 32 bit data field from the B register and shifts the data 32 bit positions to the left. The data is then shifted to the right by the number of bit positions indicated in the lowest order 5 bits of the instruction byte. The resulting data field is entered into the B register.

This operation has the effect of shifting the B register data to the left by the two's complement of the lower 5 bits of the instruction byte, with zeros filled from the right. Note that a shift count of zero (instruction 140) clears the B register.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 4 clock periods

Instruction 200-217

Machine Code	Instruction Mnemonics		Description
200-217	LA4	constant	Enter A with 4 bit constant

This instruction reads the lowest order four bits from the instruction byte and enters this value as a positive integer into the A register.

Issue Conditions:

- A register free

Completion Times:

- Next instruction in issue position in 1 clock period
- A register is free in 3 clock periods

Instruction 220-237

Machine Code	Instruction Mnemonics		Description
220-237	LA12	constant	Enter A with 12 bit constant

This instruction utilizes the lowest order four bits from the instruction byte and the eight bits of data from the next byte in the instruction stream to create a 12 bit constant. (The low order 4 bits of the instruction byte will be the high order four bits of the constant.) The resulting 12 bit positive integer is entered into the A register.

Issue Conditions:

- A register free

Completion Times:

- Next instruction in issue position in 2 clock periods
- A register is free in 3 clock periods

Instruction 240-257

Machine Code	Instruction Mnemonics	Description
240-257	LAM label	Enter A from constant local memory address

This instruction utilizes the lowest order four bits from the instruction byte and the eight bits of data from the next byte in the instruction stream to create a 12 bit address. (The low order 4 bits of the instruction byte will be the high order four bits of the address.) The resulting 12 bit positive integer is used as a local memory address. The content of the local memory address is read and the data entered into the A register.

Issue Conditions:

- A register free
- Local memory free

Completion Times:

- Next instruction in issue position in 2 clock periods
- A register is free in 6 clock periods
- Local memory is free in 4 clock periods

Instruction 260-277

Machine Code	Instruction Mnemonics		Description
260-277	SAM	label	Store A into constant local memory address

This instruction utilizes the lowest order four bits from the instruction byte and the eight bits of data from the next byte in the instruction stream to create a 12 bit address. (The low order 4 bits of the instruction byte will be the high order four bits of the address.) The resulting 12 bit positive integer is used as a local memory address. The content of the A register is read and this data stored into local memory.

Issue Conditions:

- A register free
- Local memory free

Completion Times:

- Next instruction in issue position in 2 clock periods
- A register is free in 1 clock period
- Local memory is free in 4 clock periods

Instruction 300-317

Machine Code	Instruction Mnemonics		Description
300-317	LB4	constant	Enter B with 4 bit constant

This instruction reads the lowest order four bits from the instruction byte and enters this value as a positive integer into the B register.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 1 clock period
- B register is free in 3 clock periods

Instruction 320-337

Machine Code	Instruction Mnemonics		Description
320-337	LB12	constant	Enter B with 12 bit constant

This instruction utilizes the lowest order four bits from the instruction byte and the eight bits of data from the next byte in the instruction stream to create a 12 bit constant. (The low order 4 bits of the instruction byte will be the high order four bits of the constant.) The resulting 12 bit positive integer is entered into the B register.

Issue Conditions:

- B register free

Completion Times:

- Next instruction in issue position in 2 clock periods
- B register is free in 3 clock periods

Instruction 340-357

Machine Code	Instruction Mnemonics	Description
340-357	LBM label	Enter B from constant local memory address

This instruction utilizes the lowest order 4 bits from the instruction byte and the eight bits of data from the next byte in the instruction stream to create a 12 bit address. (The low order 4 bits of the instruction byte will be the high order 4 bits of the address.) The resulting 12 bit positive integer is used as a local memory address. The content of the local memory address is read and the data entered into the B register.

Issue Conditions:

- B register free
- Local memory free

Completion Times:

- Next instruction in issue position in 2 clock periods
- B register is free in 6 clock periods
- Local memory is free in 4 clock periods

Instruction 360-377

Machine Code	Instruction Mnemonics		Description
360-377	SBM	label	Store B into constant local memory address

This instruction utilizes the lowest order 4 bits from the instruction byte and the eight bits of data from the next byte in the instruction stream to create a 12 bit address. (The low order 4 bits of the instruction byte will be the high order 4 bits of the address.) The resulting 12 bit positive integer is used as a local memory address. The content of the B register is read and this data stored into local memory.

Issue Conditions:

- B register free
- Local memory free

Completion Times:

- Next instruction in issue position in 2 clock periods
- B register is free in 1 clock period
- Local memory is free in 4 clock periods

Chapter

5 Communication Channels

This chapter is a detailed description of the communications channels that interconnect the foreground processor with the background processors and the external device interfaces is presented. An extensive description of the background processor interface is also given. Detailed descriptions of the external device interfaces are provided in Appendices B, C and D.

5.1 Foreground Communication Channels

There are four communication channels in the system which link the background processors, foreground processor and peripheral equipment controllers (see Figure 5 on page 186). Most of the data traffic is between controllers and memory. Data blocks are generally 512 common memory words in length. Typical channel reservation time is approximately five microseconds.

Each channel has associated with it four background processor channel interfaces. Each background processor interface also provides common memory access for the channel. The foreground processor supervises and controls traffic on all four channels. Disk controllers are generally divided equally between the channels. The balance of interfaces on the channels can be adjusted for special system requirements.

Each channel consists of 16 data paths and four control paths. The control paths supply the timing information necessary for data gating. These 20 paths

interconnect the nodes of the channel's communication group in a continuous loop. Each node receives data on the 20 bit paths each clock period and transmits that data onward to the next node in the same clock period. In this manner data moves about the loop from any transmitting node to any receiving node.

Control messages transmitted from the foreground processor to the channel nodes are 64 bits in length, transmitted as four parcels of 16 bits each. The first parcel is inspected by each node as it is received to determine if the message is addressed to that node. The node being addressed will retrieve function and control information from the message. In any case the message will be transmitted on to the next node on the channel loop. Examples of the format of the first parcel are given on page 248. Full descriptions of the contents of the four parcel messages are given in the descriptions of interface functions later in this chapter and in Appendixes B, C and D.

5.2 Foreground Channel Structure

The foreground channel consists of 16 data lines and four control lines. The 20 signals are transmitted around a loop which consists of the foreground processor, one or more background processors and one or more peripheral equipment interfaces. Each of these nodes in the foreground channel loop receives the 20 signals from the preceding node and relays the data to the following node. The signals may be altered as they pass a node when that node has been addressed by the foreground processor.

The foreground processor manages the foreground channel by directing function instructions to the other nodes on the channel. Each node recognizes a unique identification code and stores parameters associated with the function for further use. Nodes other than the foreground processor may continue a dialog over the channel once they have been properly initiated. The following four control signals implement this communication.

5.2.1 Channel Function Pulse

This signal originates at the foreground processor and proceeds around the foreground channel loop. The signal is a single clock period long and indicates that an foreground processor function is following on the 16 data lines. The packet of information following the function pulse consists of four 16 bit parcels which arrive at a node in the four consecutive clock periods following the function pulse. The format for data in these four parcels is as follows, beginning with the first parcel arriving after the function pulse.

16 bits - Node identification and function description

16 bits - Function response address or block length

16 bits - Upper half of foreground data word

16 bits - Lower half of foreground data word

5.2.2 Channel Response Pulse

This signal originates at a node addressed by the foreground processor and terminates at the foreground processor. The signal is a single clock period long and indicates that a two-parcel response message is following on the 16 data lines. The response occurs when the addressed node has completed the function assigned by the foreground processor. This may be only a few clock periods after the function pulse for a simple request, or it may be many clock periods and involve communication with other nodes. The response message is read by the foreground processor as a 32 bit data word. The upper half of the data word is transmitted in the first parcel and the lower half in the second parcel.

5.2.3 Channel Call Pulse

This signal originates at the foreground processor and indicates that the foreground program is ready for an interrupt request from one of the channel nodes. The channel call pulse is followed by two blank parcels of data on the 16 data lines as the call pulse leaves the foreground processor. Any node which has an interrupt request replaces the two blank parcels with its interrupt request data. If several nodes have simultaneous requests the last such node is the one which will receive foreground processor attention. When the call pulse has traveled around the channel loop and arrives back at the foreground processor the following two parcels of data are read as a 32 bit data word.

5.2.4 Channel Data Pulse

This signal travels between two nodes on the channel loop to coordinate data transfers. The foreground processor is not involved with this control line and simply relays the signal and the 16 data bits as a transparent node. The channel data pulse is only used when two nodes have been initiated by the foreground processor -- one as a data transmitter and one as a data receiver. The transmitting node begins a dialog with a channel data pulse, followed by a stream of data on the 16 data lines. The receiving node recognizes the data pulse and captures the passing data stream. The receiving node responds with a channel data pulse to the transmitting node when the data has been processed.

The sequence of data transmission and response may be continued until the functions requested by the foreground processor have been completed. Both the transmitting node and the receiving node must have the parameters

necessary for determining block length from the initiating functions of the foreground processor. Both nodes break the relay path of the channel data pulse to prevent that pulse from continuing around the channel loop.

The receiving node in such a data transfer pair is initiated first by the foreground processor. This node sends a function response as soon as the parameters have been captured. The transmitting node is initiated second, and this node does not send a function response until the entire data transfer sequence has been completed.

5.3 Foreground Channel Programming

The foreground processor is responsible for the supervision of the foregound system's four channels. The majority of the foreground processor time is spent in a scanning loop doing the following functions.

1. Testing channel zero for busy
2. Testing channel one for busy
3. Testing channel two for busy
4. Testing channel three for busy
5. Testing Real Time Clock (RTC) interval limit

When a channel is found idle the foreground processor program initiates a call for interrupt on that channel. It remembers this mode for the channel, and when the channel is idle again, the program evaluates the interrupt call response.

An interrupt call response presents the foreground processor program with a byte address for the channel sequence required. This sequence is initiated, and the following is an example of the exchange which takes place over the channel for a disk controller interrupt with data for common memory.

STEP ONE:

The foreground processor sends a function to the background processor interface to prepare the interface to receive a block of disk data. The function is immediately acknowledged with a response signal.

STEP TWO:

The foreground processor sends a function to the disk controller to begin the data transmission. The function response is delayed until the operation is completed.

STEP THREE:

The disk controller sends a data pulse to the background processor interface and follows this with a block of 256 parcels of data. The background processor interface loads this data into its internal buffer and then initiates a common memory write reference. When the data has been written into memory the background processor interface sends a data pulse to the disk controller.

Step three is repeated eight times to transfer one sector of disk data into common memory.

STEP FOUR:

The disk controller receives the eighth data pulse from the background processor interface and then sends a response pulse to the foreground processor. This releases the channel for other activity.

5.4 Foreground Communication Channel Interface Descriptions

5.4.1 Foreground Channel Loop Interfaces

The four foreground channel loops provide the basis for control of the background processors by the foreground processor and transferring data between common memory, mass storage device controllers and network interfaces. Four background processors connect to each of the channel loops. Additional device controllers for interfacing mass storage devices, other computer systems and networks may be connected to any channel loop.

An interface recognizes functions being directed to it from the foreground processor by decoding fields in the first parcel of data associated with a channel function pulse. Bits 15, 14 and 13 of the function parcel identify the interface type and delineate the format of the rest of the parcel.

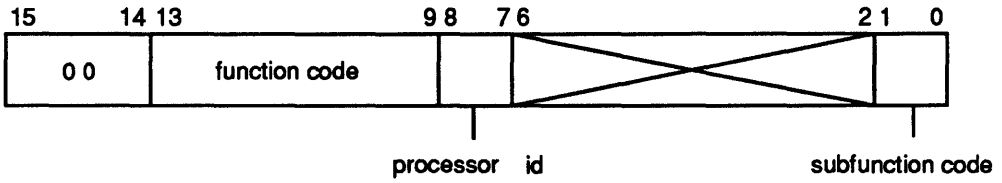
The format of the function parcel follows for the various interfaces:

5.5 Background Processor and Common Memory Interface

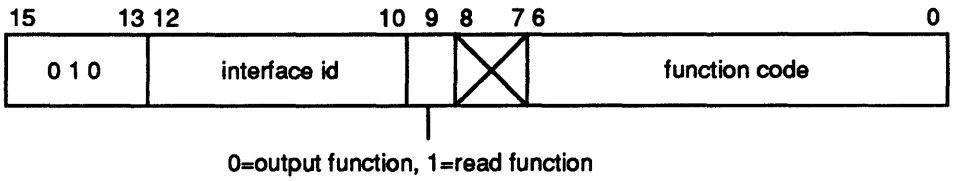
Each of the 16 background processors has a channel interface into one of the four channels in the foreground system. There are four background processor ports per channel. This interface is the vehicle for the foreground processor to

control the operation of the background processor. Each background processor interface also provides access to common memory.

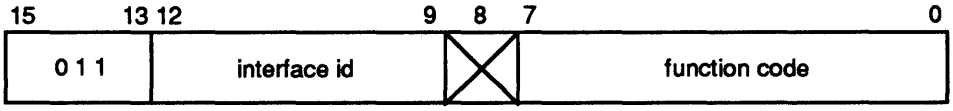
Background Processor Channel Interface



Low Speed Interface



Disk Controller Interface (DD40 or DD49)



5.5.1 Channel Call Response Register

This register is entered with 16 bits of channel function data, referred to as the response address, when certain foreground processor function requests (functions 10, 12 and 17) are performed. The parcel of data is held until replaced by another value from a new function request.

The background processor status register and error data register may generate an foreground processor interrupt request as a result of certain bit selections in those registers. When an interrupt request condition exists, the foreground interface will respond to a foreground channel call by entering the content of the channel call response register on the channel data lines following the channel call pulse.

5.5.2 Channel Common Memory Access

Each background processor interface provides access to common memory. The background processor common memory requests have priority over the requests of the foreground system. The foreground processor makes common memory requests through one of the background processor interfaces for those

foreground devices on the same channel. The priorities for the common memory requests are as follows.

1. Background processor operand references
2. Background processor instruction references
3. Foreground channel transfer references

5.5.3 Foreground Reference Registers

There are two foreground reference registers. These registers are 32 bits in length and provide a vehicle for the foreground processor to communicate with the common memory. One register is associated with the upper half of a 64 bit word. The other register is associated with the lower half of the word.

The foreground processor can read and enter these registers individually through channel function requests. The registers are used together in reading 64 bit words from the foreground memory buffer or storing words into the buffer.

5.5.4 Foreground Memory Buffer

The foreground memory buffer provides 64 words of 64 bit length for assembling and disassembling information being read from or entered into common memory.

Use of the foreground memory buffer is often implicit in foreground channel functions such as Load Background Registers (function 10). The four words to be loaded into the Program (P) address register, status register, base register, and limit register are actually read from common memory into the foreground memory buffer and transferred to the registers from there.

The foreground memory buffer can also be used explicitly, such as to hold the 64 bit words the foreground processor is reading from common memory while the foreground processor disassembles them into 32 bit blocks and reads them over the channel.

5.5.5 Common Memory Reference Address Register

The memory port contains a 32 bit common memory reference address register which is entered from the channel data word when a function 10, 11, 17, 20, 21, 26, 30, 31 or 36 from the foreground executes. The address is used for common memory read and write references.

5.5.6 Common Memory Reference Length Register

The background processor channel interface contains a 10 bit common memory reference length register. The register is entered from the lower 10 bits of the channel function word when a foreground function 20 or 30 executes. The value in the register is the number of 64 bit words read from or written to common memory in a channel block data transfer. The register is entered from the low order seven bits of the channel function word when a foreground function 26 or 36 executes. The length is forced to a value of one for execution of functions 21 and 31, to a value of two for a function 11, and to a value of four for a function 10. The reference length register is enabled for entry after any of the above functions have completed execution.

5.5.7 Error Interrupts

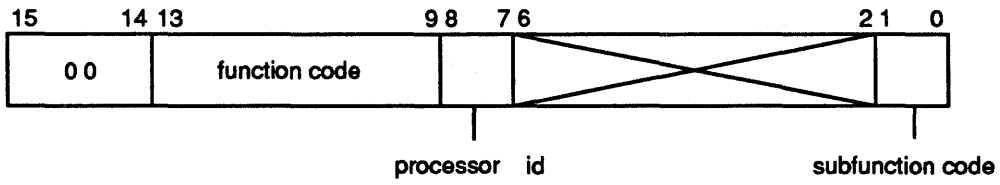
An error interrupt is generated in the background processor interface when certain bit combinations are present in the background processor status register. The error interrupt condition sends a stop pulse to the background processor. The background processor interface then waits for the idle flag to set in the status register. It then responds to future channel calls with the content of the call response address register until the status bit is cleared.

5.5.8 Background Processor Channel Interface Functions

The background processor interface to the foreground channel recognizes a unique identification code in the first data parcel following a foreground processor function pulse. When this identification code matches the background interface internal value the four parcels of data associated with the function are captured in the interface registers.

The first data parcel contains several fields which are examined by the background processor interface. These fields are listed below:

Background Processor Channel Interface



Bits 14 through 15	These two bits must be zero
Bits 09 through 13	These five bits are translated for function

Bits 07 through 08	These two bits must match the processor identification code
Bits 02 through 06	These five bits are ignored
Bits 00 through 01	These two bits are translated for subfunction

The function is identified as directed to this background processor interface when bits 07, 08, 14 and 15 match the interface values. The function and subfunction fields are then translated for the specific action requested by the foreground processor. These individual functions are listed below.

Function 10	Load background processor registers from memory
Function 11	Store background processor registers in memory
Function 12	Restart background processor program
Function 13	Stop background processor program execution
Function 14	Read program address register
Function 15	Enter program address register
Function 16	Read interface register value (subfunction)
Function 17	Enter interface register value (subfunction)
Function 20	Read common memory block to channel
Function 21	Read common memory word to foreground reference registers
Function 22	Read upper foreground reference register
Function 23	Read lower foreground reference register
Function 26	Read common memory block to foreground buffer
Function 27	Read foreground buffer word to foreground reference registers
Function 30	Store channel data block in common memory
Function 31	Store foreground reference registers in common memory word
Function 32	Enter upper foreground reference register from foreground
Function 33	Enter lower foreground reference register from foreground
Function 36	Store foreground buffer data in common memory
Function 37	Store foreground reference registers in foreground buffer word

5.5.8.1 Function 10 - Load Background Processor Registers from Memory

This function loads the background processor P register, status register, Base Address (BA) register, and Limit Address (LA) register from four consecutive locations in common memory. The background processor program is then initiated by forcing a branch to the address in the P register.

The four registers entered in this function are each 32 bits in length. The data is taken from the lower half of each of the four words read from common memory. The background processor is assumed to be idle at the time this function is performed. This is a responsibility of the foreground processor program.

The second parcel of function data is captured in the call response register. The third and fourth parcels are then assembled and entered in the common memory reference address register. A common memory read reference is then initiated with a four-word block length. The four words of common memory data are read into the foreground memory buffer. This data is then delivered to the background processor and background program execution is initiated. A channel response pulse is then sent to the foreground processor with a blank data word. The'valid field bit' associated with each field of the Instruction Stack is invalidated.

5.5.8.2 Function 11 - Store Background Processor Registers in Memory

This function stores the current background processor P register and status register values in two consecutive locations in common memory. The background processor program is not interrupted in this process.

The two register values stored in common memory are each 32 bits in length. The data is entered in the bottom half of the memory word with zero fill in the upper half. The function is executed by assembling the third and fourth parcels of function data and entering this 32 bit value in the common memory reference address register. The current program address value and the status register value are then entered in the foreground memory buffer. A common memory write reference is initiated with a two-word block length. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.3 Function 12 - Restart Background Processor Program

Function 12 restarts the background processor program at the program address currently in the P register. The background processor was assumed to be idle at the time of the foreground function request.

The second parcel of function data is captured in the call response register. The background processor program is then restarted at the current program address. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.4 Function 13 - Stop Background Processor Program Execution

This function stops the background processor program at the current program address. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.5 Function 14 - Read Program Address Register

Read program address register reads the current content of the background processor pP register. A channel response pulse is sent to the foreground processor with the program address value in the data word. The background processor program is not interrupted in this process.

5.5.8.6 Function 15 - Enter Program Address Register

This function assembles the third and fourth parcels of the function packet data and enters this 32 bit value in the background processor P register. The background processor is assumed to be idle during this process. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.7 Function 16 - Read Interface Register Value

Function 16 reads the content of one of the interface registers. A channel response pulse is sent to the foreground processor with the register value in the data word. The two-bit subfunction code in the first parcel of the function packet data is translated to select which register value to read. The translations are interpreted as follows:

Subfunction 0 - Read background processor status register

Subfunction 1 - Read common memory reference address register

Subfunction 2 - Read background processor error data register

5.5.8.8 Function 17 - Enter Interface Register Value

The second parcel of the function data is captured in the call response register. This function then assembles the third and fourth parcels of the function packet data and enters this 32 bit value in one of the interface registers. A channel response pulse is sent to the foreground processor with a blank data word. The two-bit subfunction code in the first parcel of the function packet data is translated to select which interface register to enter. The translations are interpreted as follows:

Subfunction 0 - Enter background processor status register

Subfunction 1 - Enter common memory reference address register

Subfunction 2 - Enter background processor error data register

5.5.8.9 Function 20 - Read Common Memory Block to Channel

Function 20 reads a block of data from common memory and transmits the data over the foreground channel to another channel node. If the block length is greater than 64 common memory words the transmission is broken up into 64 word segments and transmitted in several sections.

The second parcel of function data is captured and held in the common memory reference length register. This data is limited to 10 bits in length and specifies the common memory block length. The third and fourth parcels of the function data packet are then assembled and entered in the common memory reference address register. A common memory read reference is then initiated using the block length specified. If the block length is greater than 64 words, a 64 word block is read. This data is assembled in the foreground memory buffer.

A channel data pulse is then sent to the receiving node, followed by the 64 words (or less) in the foreground memory buffer. The foreground interface then waits for a channel data pulse from the receiving node to indicate that the block of data has been processed. If the channel block length was less than 64 words the function is complete and a channel response pulse is sent to the foreground processor with a blank data word.

If the channel block length was 64 words a further common memory reference is initiated. The block length is reduced by 64 counts. A copy of the common memory address is increased by 64 counts. If the channel block length is now zero the function is complete and a channel response pulse is sent to the foreground processor with a blank data word. Otherwise the above sequence is repeated and another section of the requested data is transferred over the channel.

5.5.8.10 Function 21 - Read Common Memory Word to the Foreground Reference Registers

This function reads one word from common memory and enters the data in the two foreground reference registers.

The third and fourth parcels of the function data packet are assembled and entered in the common memory reference address register. A common memory read reference is then initiated using a one-word block length. This word is read to the foreground memory buffer. The upper and lower halves of this word are then entered in the corresponding reference registers. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.11 Function 22 - Read Upper Foreground Reference Register

Ffunction 22 reads the content of the upper foreground reference register. A channel response pulse is sent to the foreground processor with the reference register value in the data word.

5.5.8.12 Function 23 - Read Lower Foreground Reference Register

Function 23 reads the content of the lower foreground reference register. A channel response pulse is sent to the foreground processor with the reference register value in the data word.

5.5.8.13 Function 26 - Read Common Memory Block to Foreground Memory Buffer

This function reads a block of data from common memory into the foreground memory buffer.

The second parcel of function data is captured and held in the common memory reference length register. This data is limited to 7 bits in length and specifies the common memory block length. The third and fourth parcels of the function data packet are assembled and entered into the common memory reference address register. A common memory read reference is then initiated using the block length specified. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.14 Function 27 - Read Foreground Memory Buffer Word to Foreground Reference Registers

This function addresses one word currently in the foreground memory buffer and enters that data in the two foreground reference registers.

The second parcel of function data is captured and held. This data is used as a buffer pointer to read one word to the foreground reference registers. The upper and lower halves of this word are entered in the corresponding reference registers. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.15 Function 30 - Store Channel Data Block in Common Memory

Function 30 reads a block of data from another node on the foreground channel and stores that data in common memory. If the block length is greater than 64 common memory words the transmission is broken up into 64 word segments and transmitted in several sections.

The second parcel of function data is captured and held in the common memory reference length register. This data is limited to 10 bits in length and specifies the common memory block length. The third and fourth parcels of the function data packet are then assembled and entered in the common memory reference address register. The foreground interface then sends a channel response pulse with a blank data word and waits for a channel data pulse from the transmitting node.

The channel data pulse signals the arrival of a block of data which is loaded into the foreground memory buffer. A common memory write reference is then initiated using the block length specified. If the block length is greater than 64 words, a 64 word block is written.

A channel data pulse is then sent to the transmitting node. If the channel block length was less than 64 words the function is complete and the interface returns to a normal channel idle mode with no function response.

If the channel block length was 64 words the block length is reduced by 64 counts. A copy of the common memory address is increased by 64 counts. If the channel block length is now zero the function is complete and the interface returns to a normal channel idle mode with no function response. Otherwise the above sequence is repeated and another section of data is transferred over the channel.

5.5.8.16 Function 31 - Store Foreground Reference Registers in Common Memory Word

This function writes one word into common memory using the data in the two foreground reference registers.

The third and fourth parcels of the function data packet are assembled and entered in the common memory reference address register. The first word in the foreground memory buffer is then loaded from the foreground reference registers. A common memory write reference is initiated using a one-word block length. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.17 Function 32 - Enter Upper Foreground Reference Register from Foreground

Function 32 enters the upper foreground reference register with 32 bits of data from the foreground processor. The data is assembled from the third and fourth parcels of the function packet data. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.18 Function 33 - Enter Lower Foreground Reference Register from Foreground

Function 33 enters the lower foreground reference register with 32 bits of data from the foreground processor. The data is assembled from the third and fourth parcels of the function packet data. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.19 Function 36 - Store Foreground Buffer Data in Common Memory

This function writes a block of data in common memory from the foreground memory buffer.

The second parcel of function data is captured and held in the common memory reference length register. This data is limited to 7 bits in length and specifies the common memory block length. The third and fourth parcels of the function data packet are assembled and entered in the common memory reference address register. A common memory write reference is then initiated using the block length specified. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.8.20 Function 37 - Store Foreground Reference Registers in Foreground Buffer Word

This function assembles a 64 bit word from the data in the foreground reference registers and enters that word in a specific address in the foreground memory buffer.

The second parcel of function data is captured and held. This data is used as a buffer pointer to enter one word from the foreground reference registers. The upper and lower halves of this word are entered from the corresponding reference registers. A channel response pulse is sent to the foreground processor with a blank data word.

5.5.9 Background Processor Status Register

The background processor status register contains a collection of 32 status flags for the background processor. Individual bits in this register indicate mode and condition of the background processor program. The significance of the individual bits is summarized below.

Bit 31	System mode flag
Bit 30	Interrupt on instruction stack parity error
Bit 29	Instruction stack parity error

Bit 28	Semaphore flag value
Bit 27	Semaphore pointer bit four (16 cpus), bit three (1-8 cpus)
Bit 26	Semaphore pointer bit three (16 cpus), bit two (1-8 cpus)
Bit 25	Semaphore pointer bit two (16 cpus), bit one (1-8 cpus)
Bit 24	Semaphore pointer bit one (16 cpus), bit zero (1-8 cpus)
Bit 23	Semaphore pointer bit zero (16 cpus)
Bit 22	Background processor idle
Bit 21	Enable real time clock entry
Bit 20	Interrupt on reciprocal table error
Bit 19	Reciprocal table error
Bit 18	Enable floating-point range tests
Bit 17	Interrupt on floating-point range error
Bit 16	Floating-point range error
Bit 15	Interrupt on common memory range error
Bit 14	Common memory write range error
Bit 13	Common memory read range error
Bit 12	Interrupt on local memory parity error
Bit 11	Local memory parity error bit 48-63
Bit 10	Local memory parity error bit 32-47
Bit 09	Local memory parity error bit 16-31
Bit 08	Local memory parity error bit 00-15
Bit 07	Interrupt on background program exit
Bit 06	Background program exit flag
Bit 05	Exit parameter bit five
Bit 04	Exit parameter bit four
Bit 03	Exit parameter bit three
Bit 02	Exit parameter bit two
Bit 01	Exit parameter bit one
Bit 00	Exit parameter bit zero

5.5.9.1 Status Register Bit 31 - System Mode Flag

The system mode flag bit is used by the foreground processor to indicate the character of the program currently running on this background processor. The bit does not interact with any hardware in the background processor.

5.5.9.2 Status Register Bit 30 - Interrupt on Instruction Stack Parity Error

This bit is set by the foreground processor to cause an interrupt of the foreground program when bit 29 of the status register is set. The foreground interface will respond to a channel call when bits 29 and 30 are both set in the status register.

5.5.9.3 Status Register Bit 29 - Instruction Stack Parity Error

Bit 29 is set by the background processor if a parity error occurs in the operation of the background processor instruction stack. Eight bits of error status data are captured in the error data register at the time of occurrence. The foreground interface will respond to a channel call when bits 29 and 30 are both set in the status register.

5.5.9.4 Status Register Bit 28 - Semaphore Flag Value

This bit is set and cleared by actions of the background processor program which alter the assigned semaphore flag. Actions of other background processors assigned to this same semaphore flag do not alter the value in this background processor's status register.

5.5.9.5 Status Register Bits 23 through 27 - Semaphore Pointer (16 cpus)

These five bits are used by the foreground processor to assign one of the 32 semaphore flags to this background processor. Background processor instructions then interact with this assigned semaphore flag and are not able to alter the other 31 semaphore flags.

For systems with one through eight processors the status register uses bits 24 through 27, yielding a total of 16 semaphore flags.

5.5.9.6 Status Register Bit 22 - Background Processor Idle

The background processor idle bit provides a monitor of the background processor idle status. The bit is set every clock period, beginning at deadstart if the background processor is idle. It is cleared every clock period if the background processor is active with instruction execution or memory references initiated by the background processor. An idle processor for this

purpose is one which has executed an EXIT jk instruction (000jk) or has been stopped by the foreground processor and has not yet been restarted.

5.5.9.7 Status Register Bit 21 - Enable Real Time Clock Entry

This bit enables the background processor RT S_j instruction (035-j-) to enter data in the RTC register. If this bit is cleared the RT S_j instruction executes as a pass.

5.5.9.8 Status Register Bit 20 - Interrupt on Reciprocal Table Error

Bit 20 is set by the foreground processor to cause an interrupt of the foreground program when bit 19 of the status register is set. The foreground interface will respond to a channel call when bits 19 and 20 are both set in the status register.

5.5.9.9 Status Register Bit 19 - Reciprocal Table Error

The reciprocal table error bit is set by the background processor if a parity error is detected in reading data from the reciprocal lookup table. The foreground interface will respond to a channel call when bits 19 and 20 are both set in the status register.

5.5.9.10 Status Register Bit 18 - Enable Floating-Point Range Tests

This bit is set by the foreground processor to enable the floating-point functional units to perform floating-point range tests. It may also be set or cleared by the background processor using the DFI instruction (037--2) or EFI instruction (037--3) to change the status of range testing during program execution.

5.5.9.11 Status Register Bit 17 - Interrupt on Floating-Point Range Error

Bit 17 is set by the foreground processor to cause an interrupt of the foreground program when bit 16 of the status register is set. The foreground interface will respond to a channel call when bits 16 and 17 are both set in the status register.

5.5.9.12 Status Register Bit 16 - Floating-Point Range Error

The floating-point range error bit is set when the background processor floating-point functional units detect a floating-point result which has a value out of range. The foreground interface will respond to a channel call when bits 16 and 17 are both set in the status register.

5.5.9.13 Status Register Bit 15 - Interrupt on Common Memory Range Error

This bit is set by the foreground processor to cause an interrupt of the foreground program when bit 13 or 14 of the status register are set. The foreground interface will respond to a channel call when bits 13 and 15 or bits 14 and 15 are set in the status register. The background processor may also set or clear this bit by execution of the DRI instruction (037--0) or ERI instruction (037--1).

5.5.9.14 Status Register Bit 14 - Common Memory Write Range Error

Bit 14 is set by the background processor when a common memory write reference is made with an address which exceeds the assigned range. The write function is blocked in common memory in this case. The foreground interface will respond to a channel call when bits 14 and 15 in the status register are both set. The background processor may clear this bit by execution of a DRI instruction (037--1).

5.5.9.15 Status Register Bit 13 - Common Memory Read Range Error

Bit 13 is set by the background processor when a common memory read reference is made with an address which exceeds the assigned range. The read reference returns a word of all zero bits in this case. The foreground interface will respond to a channel call when bits 13 and 15 in the status register are both set. The background processor may clear this bit by execution of a DRI instruction (037--1).

5.5.9.16 Status Register Bit 12 - Interrupt on Local Memory Parity Error

This bit is set by the foreground processor to cause an interrupt of the foreground program when any of bits 08 through 11 are set in the status register. The foreground interface will respond to a channel call if any of these four bits are set along with bit 12.

5.5.9.17 Status Register Bits 08 through 11 - Local Memory Parity Error

These bits are set in the status register when a parity error is detected in reading data from local memory in the background processor. One bit is associated with each of the background processor modules. The foreground interface will respond to a channel call if any of these four bits are set along with bit 12.

5.5.9.18 Status Register Bit 07 - Interrupt on Background Program Exit

Bit 07 is set by the foreground processor to cause an interrupt of the foreground program when bit 06 of the status register is set. The foreground

interface will respond to a channel call when bits 06 and 07 are both set in the status register.

5.5.9.19 Status Register Bit 06 - Background Program Exit Flag

The background program exit flag bit is set by the background processor when the background program executes an EXIT jk instruction (000-jk). The foreground interface will respond to a channel call when bits 06, 07 and 22 are set in the status register.

5.5.9.20 Status Register Bits 00 through 05 - Exit Parameter

This six-bit parameter is entered in the status register from the jk field of the background processor EXIT jk instruction (000jk). This parameter is then used by the foreground processor to determine the type of background program exit.

5.5.10 Background Processor Error Data Register

The error data register is a 32 bit collection of flags for errors not indicated in the background processor status register. One field of the error data register is related to the background processor instruction stack. A second field is related to errors in the operation of the common memory. The significance of individual bits is summarized below.

Bit 31	Instruction stack error upper bank, parcel zero
Bit 30	Instruction stack error upper bank, parcel one
Bit 29	Instruction stack error upper bank, parcel two
Bit 28	Instruction stack error upper bank, parcel three
Bit 27	Instruction stack error lower bank, parcel zero
Bit 26	Instruction stack error lower bank, parcel one
Bit 25	Instruction stack error lower bank, parcel two
Bit 24	Instruction stack error lower bank, parcel three
Bit 23	Memory bank address bit nine
Bit 22	Memory bank address bit eight
Bit 21	Memory bank address bit seven
Bit 20	Memory bank address bit six
Bit 19	Memory bank address bit five
Bit 18	Memory bank address bit four
Bit 17	Memory bank address bit three

Bit 16	Memory bank address bit two
Bit 15	Memory bank address bit one
Bit 14	Memory bank address bit zero
Bit 13	Not assigned
Bit 12	Enable SECDED error correction
Bit 11	Interrupt on double bit error
Bit 10	Interrupt on single bit error
Bit 09	Double bit memory error
Bit 08	Single bit memory error
Bit 07	Syndrome data bit seven
Bit 06	Syndrome data bit six
Bit 05	Syndrome data bit five
Bit 04	Syndrome data bit four
Bit 03	Syndrome data bit three
Bit 02	Syndrome data bit two
Bit 01	Syndrome data bit one
Bit 00	Syndrome data bit zero

5.5.10.1 Error Register Bits 28 through 31 - Upper Instruction Stack Error

These four bits in the error register indicate which parcel within the upper instruction stack bank caused a parity error. More than one bit in this group may be set if simultaneous errors occur.

5.5.10.2 Error Register Bits 24 through 27 - Lower Instruction Stack Error

Bits 24-37 in the error register indicate which parcel within the lower instruction stack bank caused a parity error. More than one bit in this group may be set if simultaneous errors occur.

5.5.10.3 Error Register Bits 14 through 23 - Memory Bank Address

These 10 bits in the error register indicate which common memory bank caused a single- or double-bit error.

5.5.10.4 Error Register Bit 13 - Not Assigned

The not assigned bit in the error register can be entered and read by the foreground processor but does not interact with any hardware in the background processor.

5.5.10.5 Error Register Bit 12 - Enable secded Error Correction

This bit in the error register is set by the foreground processor to enable the common memory error correction circuits. It is cleared during some types of maintenance programs to better identify failing memory bit positions.

5.5.10.6 Error Register Bits 10 and 11 - Interrupt on Common Memory Error

These two bits in the error register are set by the foreground processor to enable foreground channel call response when a single- or double-bit common memory error has occurred.

5.5.10.7 Error Register Bit 09 - Double-Bit Memory Error

Bit 09 in the error register is set when a double-bit error is detected in common memory data read to this background processor. The error may be due to a foreground channel reference through this interface or due to a background processor memory reference. A double-bit error will cause the syndrome data and bank address data to be sampled into this register. Data from a previous single-bit error will be replaced by a double-bit error. A second double-bit error will be ignored.

5.5.10.8 Error Register Bit 08 - Single-Bit Memory Error

The single-bit memory error in the error register is set when a single-bit error is detected in common memory data read by this background processor. The error may be due to a foreground channel reference through this interface or due to a background processor memory reference. A single-bit error will cause the syndrome data and bank address data to be sampled into this register unless a previous error has already loaded data.

5.5.10.9 Error Register Bits 00 through 07 - Syndrome Data

These eight bits in the error register indicate the value of the common memory syndrome data for a single- or double-bit common memory error.

Appendix

A Division and Square Root Algorithms

Division and square root are accomplished by algorithms that perform Newton's Method approximations. Support for these approximations is provided by performing parts of the algorithms in the hardware. This appendix presents the detailed development of the algorithms. The hardware support for the algorithms is also discussed in detail.

A.1 Design Considerations

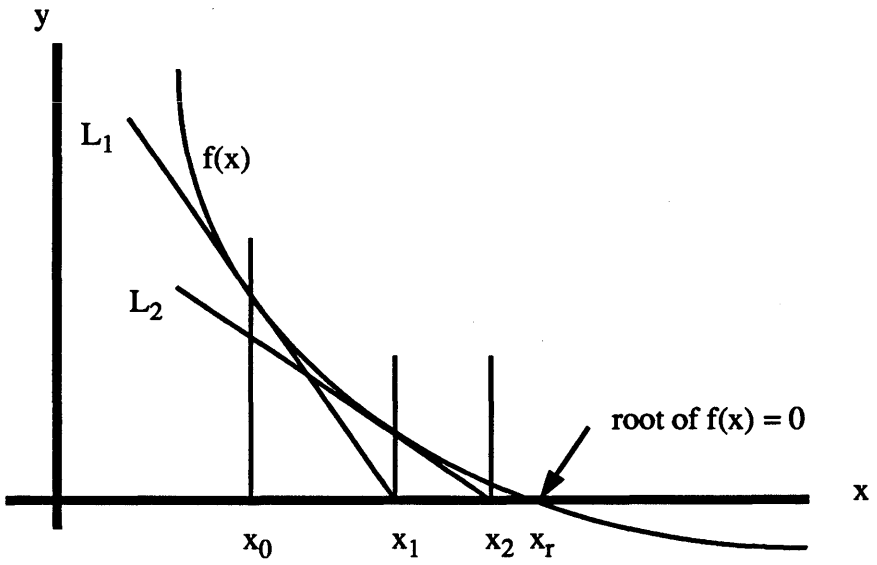
Division is the most expensive floating point operation performed by a computer in terms of both processor cycles and circuitry. Breaking a division up into a series of approximations, corrections and multiplications by applying Newton's method provides a means for limiting these expenses and provides an efficient square root as a by product.

The divide of two floating point numbers is not atomically implemented in hardware. The divide, A/B is implemented as a two step procedure of first computing 1/B and then multiplying this result by A. Therefore, only the computation of the reciprocal is implemented in hardware as the second step is an ordinary multiplication. Similarly, to implement the square root, the hardware computes the reciprocal square root, $1/\sqrt{C}$ and the square root, $\sqrt{C}$ is computed as C multiplied by $1/\sqrt{C}$. This procedure allows the computation of the square root to follow a path similar to that of the floating point divide. This appendix first describes Newton's method for an arbitrary function, then derives the steps for applying Newton's method for the reciprocal and the

reciprocal square root. Finally, it shows how the hardware is used to implement these steps.

A.2 Newton's Method

Newton's method can be used to find a root of some functions. Specifically, the problem of finding a quotient or a square root can be formulated as finding the root of a function with Newton's method. Consider the figure below:



A root of the function f(x) is defined as being a value x_r such that $f(x_r)=0$. A root is indicated graphically as the point where f(x) crosses the x axis. For a particular function f(x), and an initial approximation, x_0, of the value of x_r, a sequence of improved approximations can be calculated. Newton's method specifies calculating an improved approximation by using the root of a line (L_1) tangent to f(x) at $(x_0,f(x_0))$ as the improved approximation, x_1. X_1 is used as the basis for the next iteration. Line L_2 is defined by the point $(x_1,f(x_1))$ and the slope of f(x) at x_1. The intersection of L_2 and the x axis is another improved approximation to the root of f(x), x_2. This method can be iterated to produce a result of any desired accuracy.

The slope of any of the lines used in an approximation improvement, L_{k+1}, can be computed as the value of the derivative of f(x) evaluated at x_k, $f'(x_k)$. The slope of L_{k+1} can also be computed as $(0 - f(x_k))/(x_{k+1} - x_k)$.

- Slope of $L_{k+1} = f'(x_k)$
- Slope of $L_{k+1} = (0 - f(x_k))/(x_{k+1} - x_k)$

Equating these two formulas and solving for x_{k+1} produces:

- $x_{k+1} = x_k - f(x_k)/f'(x_k)$

The result is a formula that can be applied to a starting approximation, x_0, and iterated to produce a final approximation of the desired accuracy.

This method of successively choosing lines tangent to a curve and finding their intersection with the x axis is Newton's method. Not all functions lend themselves to having their roots calculated by Newton's method. It can be applied for the computation of the reciprocal and the reciprocal square root. From the calculus we have that Newton's method requires that the function f(x) be twice differentiable and have a continuous second derivative on the interval of interest. This is true for 1/x and $1 / x^2$ in the intervals [1/2,1] and [1/2,2] respectively, so Newton's method can be applied for these functions.

A.3 Newton's Method Applied to Reciprocal Approximation

Finding the reciprocal 1/b can be formulated as finding the root of the function F(x) = 1/x-b. The root of this function is x=1/b. Applying Newton's method, the root of function F can be computed to any accuracy by the iterative formula:

- $x_{i+1} = x_i - F(x_i)/F'(x_i)$

where F' is the first derivative of F and x_{i+1} is a better approximation to the root of F(x) than x_i.

For the reciprocal, let F(x) = 1/x - b then F(1/b) = 0 and

- $F'(x) = -1/x^2$

Substituting these formulas into the general Newton's method formula produces:

- $x_{i+1} = x_i - (1/x_i - b)/(-1/x_i^2)$

Simplifying this formula produces:

- $x_{i+1} = 2x_i - bx_i^2$ or $x_{i+1} = x_i(2 - bx_i)$

A.4 Newton's Method and the Square Root Approximation

In a manner similar to applying Newton's method to compute the reciprocal, it can be applied to compute the reciprocal square root. Finding the reciprocal square root $1/\sqrt{c}$ can be formulated as finding the root of the function $G(x) = 1/x^2 - c$. The root of this function is $x = 1/\sqrt{c}$. Applying Newton's method, the root of function G can be computed to any accuracy by the iterative formula:

- $x_{i+1} = x_i - G(x_i)/G'(x_i)$

Where G' is the first derivative of G and x_{i+1} is a better approximation to the root of G(x) than x_i.

For the reciprocal square root, let $G(x) = 1/x^2 - c$ then $G(1/\sqrt{c}) = 0$ and

- $G'(x) = -2/x^3$

Substituting these formulas into the general Newton's method formula produces:

- $x_{i+1} = x_i - (1/x_i^2 - c)/(-2/x_i^3)$

Simplifying this formula produces:

- $x_{i+1} = 3x_i/2 - cx_i^3/2$ or $x_{i+1} = x_i (3 - cx_i^2)/2 = x_i ((3 - x_i (cx_i))/2)$

The square root is produced by computing c $(1.0 / \sqrt{c})$. For any term x_{i+1}:

- $\sqrt{c} = cx_{i+1} = cx_i ((3 - x_i (cx_i))/2) = (cx_i)((3 - x_i (cx_i))/2)$

A.5 Hardware Support

Both the quotient and square root are computed by producing an initial approximation and then performing two iterations of Newton's method.

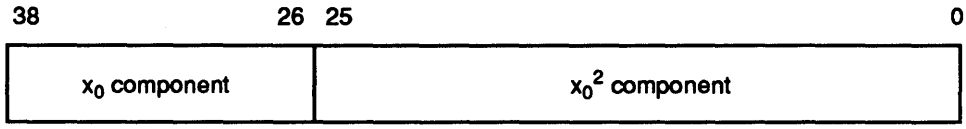
A.5.1 Reciprocal Approximation

Reciprocal approximation is performed in two distinct steps. In the first step a first approximation is formed and one iteration is completed. This step proceeds in two parts, first a table lookup using read only memory and then the terms from the lookup table are used to complete the first iteration of producing an improved approximation.

The second step is to perform another iteration, producing the reciprocal, and multiply the numerator by it to produce the quotient.

A.5.1.1 The Approximation Step of the Reciprocal

The table used in the first step of the reciprocal approximation contains 2048 words each 39 bits long. The address for the table lookup is taken from the highest 12 bits of the operand coefficient. The operand is assumed to be normalized so only 11 bits are used as the table address. These values are then used to complete the first iteration. The values retrieved by the table lookup are:



13 bits - used in the computation of $2x_0$

26 bits - used in the computation of x_0^2

Using these two constants the floating point multiply unit completes the computation of the first iteration of Newton's method:

- $x_1 = 2x_0 - bx_0^2$

The result of one iteration of Newton's method, x_1, is returned as the result of the first reciprocal approximation. The instructions that perform a first reciprocal approximation are:

Machine Code	CAL Instruction		Description
132ij-	S_i	/HS_j	Reciprocal approximation of S_j
166ij-	V_i	/HV_j	Reciprocal approximation of V_j

A.5.1.2 The Iterative Step for the Reciprocal Approximation

The second iteration of Newton's method is implemented as a sequence of code that computes a correction term from the original operand and then applies it to the first approximation to produce the second approximation:

- correction term = $(2 - bx_1)$
- $x_2 = x_1(2 - bx_1) = x_1$(correction term)

The instructions that produce the reciprocal approximation correction term are:

Machine Code	CAL Instruction		Description
126ijk	S_i	S_j*IS_k	Reciprocal iteration step 2-S_j*S_k
156ijk	V_i	V_j*IV_k	Reciprocal iteration step 2-V_j*V_k

The correction term is applied by performing a floating point multiply. The final step of the sequence is to perform a floating point multiply of the numerator by the reciprocal to produce the quotient.

A.5.1.3 Reciprocal Approximation Scalar Instruction Sequence

The sequence of CAL instructions required to produce S1/S2 follows. The vector sequence is identical, using the appropriate vector instructions.

S3	/HS2	Reciprocal approximation of S2
S4	S2*IS3	Form correction term 2-S2*S3 = 2-bx_1
S5	S3*FS4	Apply the correction term to produce S3*(2-S3*S2) = x_1(2-bx_1)
S6	S1*FS5	Produce the quotient as S1 * (1.0 / S2)

A.5.2 Square Root Approximation

Square root approximation is performed in two distinct steps. In the first step a first approximation is formed and one iteration is completed. This step proceeds in two parts, first a table lookup using read only memory and then the terms from the lookup table are used to complete the first iteration of producing an improved approximation.

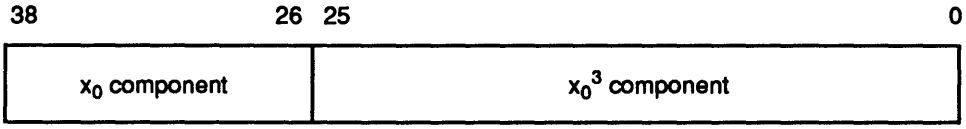
The second step performs another iteration to produce the reciprocal square root and multiplies by the original operand to produce the square root.

A.5.2.1 The Approximation Step of the Reciprocal Square Root

The table used in the first part of square root approximation contains 2048 words each 39 bits long. The address of the table lookup is taken from the least significant bit of the exponent of the operand and the 11 most significant bits of the coefficient of the operand. Of these 11 bits of the coefficient, the first bit is

discarded as the coefficient is assumed to be normalized. These values are used to complete the first iteration of producing an improved approximation.

The values retrieved from the table are:



13 bits - used in the computation of $3x_0/2$

26 bits - used in the computation of $x_0^3/2$

Using these two constants the floating point multiply unit completes the computation of the first iteration of Newton's method:

- $x_1 = (3x_0/2) - (x_0^3/2)c$

The result of one iteration of Newton's method, x_1, is returned as the result of the first reciprocal square root approximation. The instructions that perform a first reciprocal square root approximation are:

Machine Code	CAL Instruction		Description
133ij-	S_i	*QS_j	Square root approximation of S_j
167ij-	V_i	*QV_j	Square root approximation of V_j

A.5.2.2 The iterative Step for the Square Root Approximation

The second iteration of Newton's method is implemented as a sequence of code that computes the second iteration and multiplies by c to produce $\sqrt{c}$ from $1/\sqrt{c}$:

- $\sqrt{c} = c\,(1/\sqrt{c}) = cx_2 = (cx_1)\,((3 - x_1\,(cx_1))/2)$

The instructions that produce the reciprocal square root approximation iteration term are:

Machine Code	CAL Instruction		Description
127ijk	S_i	S_j*QS_k	Square root iteration $(3-S_j*S_k)/2$
157ijk	V_i	V_j*QV_k	Square root iteration $(3-V_j*V_k)/2$

The iteration term is applied by performing a floating point multiply after the first approximation has been multiplied by the original operand.

A.5.2.3 Square Root Approximation Scalar Instruction Sequence

The sequence of CAL instructions required to produce the square root of an operand in S1 follows. The vector sequence is identical, using the appropriate vector instructions.

S2	*QS1	Reciprocal square root approximation of S1. S2=x1
S3	S1*FS2	Multiply by original operand to produce S1*S2=cx_1
S4	S2*QS3	Square root iteration to produce (3-S2*S3)/2=(3-$x_1(cx_1)$)/2
S5	S3*FS4	Complete the second iteration to produce S3*S4=$(cx_1)((3-x_1(cx_1))/2)=\sqrt{c}$

Appendix

B Low-Speed Interface

The system is connected to front-end computer systems and networks by Low-Speed Interfaces that connect to one of the four communications channels. Low-speed (six megabytes per second) and high-speed (12 megabytes per second) transfer modes are available.

B.1 Low-Speed Interface

The controller communicates with the external device using two independent simplex channels -- one for input, and one for output. Each of these simplex channels provides a 16 bit data path and control signals associated with each channel which include Data Ready, Resume, Disconnect and Master Clear. The channels can transfer data in either low-speed mode at six megabytes per second or high-speed mode at 12 megabytes per second. A status register, call response register, transfer length register and a 512 64 bit word buffer are associated with each of the simplex channels.

B.2 General Description

B.2.1 Interface Functions

The LSX interface responds to 13 function requests from the Foreground Processor (FP). These are listed below in octal notation.

Function 00	Read input status
Function 01	Master clear device
Function 02	Begin low-speed mode
Function 03	Clear interface flags
Function 04	Disable input interrupt
Function 05	Enable input interrupt
Function 06	Receive N words from device
Function 07	Transfer N words to memory
Function 10	Transfer N words from memory
Function 11	Send N words to device with disconnect
Function 12	Send N words to device no disconnect
Function 13	Send disconnect to device
Function 14	Read output status

B.2.1.1 Function 00 - Read Input Status

This function causes an immediate response with 32 bits of interface status data. The status data includes both input and output flags and is the same data as is provided for a function 14 response. The input interrupt flag is cleared by this function execution. Bit assignments in the status response are as follows.

Bit 00 - 09	Input remaining length
Bit 10	Input word waiting
Bit 11	Input disconnected
Bit 12	Early input disconnect
Bit 13	Output resume error
Bit 14	Input buffer lower parity error

Bit 15	Input buffer upper parity error
Bit 16s - 25	Output remaining length
Bit 26	Output buffer lower parity error
Bit 27	Output buffer upper parity error
Bit 28	Input data first four bit error
Bit 29	Input data second four bit error
Bit 30	Input data third four bit error
Bit 31	Input data fourth four bit error

B.2.1.2 Function 01 - Master Clear Device

This function sends a continuous master clear signal to the external device over both the input path and the output path. The last three parcels of the function packet are not used. An immediate null response is sent to the FP. The master clear signal remains present until a function 03 is executed or the entire system is dead started.

B.2.1.3 Function 02 - Begin Low-Speed Mode

Function 02 sets a low-speed mode flag in the interface. The last three parcels of the function packet are not used. An immediate null response is sent to the FP. Following functions involving data transfers to or from the external device are executed in the low-speed mode. The low-speed mode flag remains set until a function 03 is executed or the entire system is dead started.

B.2.1.4 Function 03 - Clear Interface Flags

This function clears all mode flags in the interface circuits. The last three parcels of the function packet are not used. An immediate null response is sent to the FP. This function will abort a data transfer to or from the external device if one is in process.

B.2.1.5 Function 04 - Disable Input Interrupt

Disable Input Interrupt clears the interrupt enable flag in the interface. The last three parcels of the function packet are not used. An immediate null response is sent to the FP.

B.2.1.6 Function 05 - Enable Input Interrupt

This function sets the interrupt enable flag in the interface. The second parcel of the function packet is captured in the input interrupt register. The last two parcels of the function packet are not used. An immediate null response is sent to the FP. The interrupt enable flag, together with an input word waiting flag, causes a call response with the content of the input interrupt register.

B.2.1.7 Function 06 - Receive N Words From Device

Function 06 sets the device input to an active mode. The second parcel of the function packet is captured in the input interrupt register. The third parcel of the function packet is not used, and the fourth parcel of the function packet is captured in the input length register. The interrupt enable flag is cleared. An immediate null response is sent to the FP.

The external device may have sent one to four parcels of data to the interface prior to the execution of this function. If this is the case, these parcels are included in the loading of the input buffer with device data. The input buffer is loaded until the length count has been satisfied or until a disconnect is received from the external device.

When loading of the input buffer is complete, the interface will respond to a foreground channel call with the contents of the input interrupt register.

B.2.1.8 Function 07 - Transfer N Words to Memory

Function 07 dumps the content of the input buffer over the foreground channel to Common Memory (CM). The second and third parcels of the function packet are not used. The fourth parcel of the function packet is captured in the input length register. A null response is sent to the FP when the transfer has been completed.

The CM access in the Background Processor (BP) must be ready to receive the buffer data when this function is executed. This condition is established by a previous foreground channel function. The transfer lengths must be the same in the two foreground functions.

B.2.1.9 Function 10 - Transfer N Words From Memory

This function prepares the interface output buffer to receive data from CM. The second and third parcels of the function packet are not used. The fourth parcel of the function packet is captured in the output length register, and an immediate null response is sent to the FP.

No further action will occur until a foreground function is sent to the BP, which will send the data over the foreground channel. The transfer lengths must be the same in the two foreground functions.

B.2.1.10 Function 11 - Send N Words to Device With Disconnect

Function 11 sets the device output to an active mode. The second parcel of the function packet is captured in the output interrupt register. The third parcel of the function packet is not used. The fourth parcel of the function packet is captured in the output length register. An immediate null response is sent to the FP.

The output buffer data is then sent to the device until the output length has been satisfied. A disconnect signal is sent to the device. The interface then responds to a FP call with the content of the output interrupt register.

B.2.1.11 Function 12 - Send N Words to Device No Disconnect

This function sets the device output to an active mode. The second parcel of the function packet is captured in the output interrupt register. The third parcel of the function packet is not used. The fourth parcel of the function packet is captured in the output length register. An immediate null response is sent to the FP.

The output buffer data is then sent to the device until the output length has been satisfied. No disconnect signal is sent to the device. The interface then responds to a FP call with the content of the output interrupt register.

B.2.1.12 Function 13 - Send Disconnect to Device

Function 13 sends a disconnect signal to the device over the output path. The second, third and fourth parcels of the function packet are not used. An immediate null response is sent to the FP. This function will abort an active output mode if one is in process.

B.2.1.13 Function 14 - Read Output Status

Read Output Status causes an immediate response with 32 bits of interface status data. The status data includes both input and output flags and is the same data as is provided for a function 00 response. The output interrupt flag is cleared by this function execution.

B.2.2 Foreground Channel Structure

The foreground channel consists of 16 data lines and four control lines. The 20 signals are transmitted around a loop which consists of the FP, one or more BPs and one or more peripheral equipment interfaces. Each of these nodes in the foreground channel loop receives the 20 signals from the preceding node and relays the data to the following node. The signals may be altered as they pass a node that has been addressed by the FP.

The FP manages the foreground channel by directing function instructions to the other nodes on the channel. Each node recognizes a unique identification code and stores parameters associated with the function for further use. Nodes other than the FP may continue a dialog over the channel once they have been properly initiated. The four control signals which implement this communication are as follows.

B.2.2.1 Channel Function Pulse

This signal originates at the FP and proceeds around the foreground channel loop. The signal is a single clock period long and indicates that a FP function is following on the 16 data lines. The packet of information following the function pulse consists of four 16-bit parcels which arrive at a node in the four consecutive clock periods following the function pulse. The format for data in these four parcels is as follows, beginning with the first parcel arriving after the function pulse:

16 bits) Node identification and function description

16 bits) Function response address or block length

16 bits) Upper half of foreground data word

16 bits) Lower half of foreground data word

B.2.2.2 Channel Response Pulse

This signal originates at a node addressed by the FP and terminates at the FP. The signal is a single clock period long and indicates that a two-parcel response message is following on the 16 data lines. The response occurs when the addressed node has completed the function assigned by the FP. This may be only a few clock periods after the function pulse for a simple request, or it may be many clock periods and involve communication with other nodes. The response message is read by the FP as a 32-bit data word. The upper half of the data word is transmitted in the first parcel and the lower half in the second parcel.

B.2.2.3 Channel Call Pulse

This signal originates at the FP and indicates that the foreground program is ready for an interrupt request from one of the channel nodes. The channel call pulse is followed by two blank parcels of data on the 16 data lines as the call pulse leaves the FP. Any node which has an interrupt request replaces the two blank parcels with its interrupt request data. If several nodes have simultaneous requests, the last such node is the one which will receive FP attention. When the call pulse has traveled around the channel loop and arrives back at the FP, the following two parcels of data are read as a 32-bit data word.

B.2.2.4 Channel Data Pulse

This signal travels between two nodes on the channel loop to coordinate data transfers. The FP is not involved with this control line and simply relays the signal and the 16 data bits as a transparent node. The channel data pulse is only used when two nodes have been initiated by the FP -- one as a data transmitter, and one as a data receiver. The transmitting node begins a dialog with a channel data pulse, followed by a stream of data on the 16 data lines. The receiving node recognizes the data pulse and captures the passing data stream. The receiving node responds with a channel data pulse to the transmitting node when the data has been processed.

The sequence of data transmission and response may be continued until the functions requested by the FP have been completed. Both the transmitting node and the receiving node must have the parameters necessary for determining block length from the initiating functions of the FP. Both nodes break the relay path of the channel data pulse to prevent that pulse from continuing around the channel loop.

The receiving node in such a data transfer pair is initiated first by the FP. This node sends a function response as soon as the parameters have been captured. The transmitting node is initiated second, and this node does not send a function response until the entire data transfer sequence has been completed.

B.2.3 Interface Circuit Descriptions

The following descriptions identify individual detail circuit parameters and characteristics of the interface circuits in this board stack.

B.2.3.1 Foreground Channel Function Format

The FP addresses this node through the first parcel of a four- parcel function request. The format for the 16 bits in this parcel is as follows:

Bit 00 - 03 Function specification

Bit 04 - 09	Not used
Bit 10 - 12	Node identification (plugged)
Bit 13	Constant zero
Bit 14	Constant one
Bit 15	Constant zero

B.2.3.2 Call Response Parameters

The second parcel in the FP function packet is used for the call response parameter for some interface functions. The interface has a 16-bit register to hold the device input call response and a separate 16-bit register to hold the device output call response. When a call response is made the upper parcel of the response data is always zero, and the lower parcel contains the appropriate 16-bit register content.

B.2.3.3 Data Length Parameters

The fourth parcel in the FP function packet is used for defining the data block length for some interface functions. The length is limited to the lowest order 10 bits in the parcel. The upper six bits are ignored.

The interface has a 10-bit block length register for the input buffer and a separate 10-bit block length register for the output buffer. These lengths define the number of 64-bit words to be transferred in executing the function.

In the case of data transfers to or from the CM the data block size moved in a single foreground channel unit is limited to 64 words (256 parcels). If the block length specified in the function is larger than this value, the block is broken up by the transmitting and receiving node hardware and sent in 64- word units until the remainder is less than 64. This process is transparent to the FP program. A total length of zero is not allowed and will cause the foreground channel to hang up.

B.2.3.4 Input (or Output) Remaining Length

The status readout to the foreground program includes the remaining length in the input and output block length registers. These registers are loaded with the originally specified block length and are then decremented by one count as each 64-bit word is moved to or from the interface buffer.

B.2.3.5 Input Word Waiting Flag

This interface flag is set when the input mode is not active and the external device sends a word. In this situation the interface does not respond with a resume pulse and the device transmitter is left waiting. If the enable interrupt flag is set, the interface responds to a foreground call to bring the attention of the foreground program to the arriving data. The input word waiting flag is cleared by execution of a function 03 or 06.

B.2.3.6 Input Disconnected Flag

The input disconnect flag is set when an input disconnect signal arrives from the external device. The flag is cleared when a function 00 or 03 is executed.

B.2.3.7 Early Disconnect Flag

This interface flag is set when an input disconnect signal arrives and the interface was not expecting it. It is expected only during an input mode active and a remaining length count of zero. The flag is cleared when a function 00 or 03 is executed.

B.2.3.8 Output Resume Error Flag

The output resume error flag is set when an output resume signal arrives and the interface was not expecting it. It is expected during an output mode active when a ready pulse has been sent. The flag is cleared when a function 03 or 13 is executed.

B.2.3.9 low-speed Mode Flag

This interface flag is set when a function 02 is executed. Communication with the external device is in a low-speed mode until a function 03 is executed.

B.2.3.10 Enable Interrupt Flag

The enable interrupt flag provides an optional foreground call response. The enable interrupt flag is set by execution of a function 05. It enables the setting of the input interrupt flag when a word waiting flag or a disconnect flag are also set. It is cleared by the setting of the input interrupt flag or by execution of a function 03, 04 or 06.

B.2.3.11 Input Interrupt Flag

This flag is set when the interface input circuits wish to respond to a foreground channel call. If both the input and output interrupt flags are set, the input interrupt flag has priority in responding to the foreground channel call.

This flag is set when the input word waiting flag (or input disconnected flag) and the enable interrupt flag are both set. It is also set when execution of a function 06 has been completed. It is cleared on execution of a function 00, 03, 04, 05, 06 or 07.

B.2.3.12 Output Interrupt Flag

The output interrupt flag is set when the interface output circuits wish to respond to a foreground channel call. If both the input and output interrupt flags are set, the input interrupt flag has priority in responding to the foreground channel call.

This flag is set when the execution of a function 11 or 12 has been completed. It is cleared on execution of a function 03, 10, 11, 12, 13 or 14.

B.2.3.13 System Dead Start

A system dead start is treated by the interface circuits as if a function 03 had been executed. In addition, the external device master clear signals are sent over both the input and output paths for the duration of the system dead start signal.

CRAY-3 Low-Speed Controller Functions

Fcn	Description	A Register		B Register		Function Response		Respond to call after:
		Parcel-0	Parcel-1	Parcel-2	Parcel-3	Parcel-0	Parcel-1	
Common Functions:								
000	Read Input Status	010aaa- -----0000	–	–	–	Status-high	Status-low	N/A (clears input interrupt)
001	Set Output Master Clear	010aaa- -----0001	–	–	–	0	0	N/A
002	Begin Low-Speed Mode	010aaa- -----0010	–	–	–	0	0	N/A
003	Clear Interface Flags	010aaa- -----0011	–	–	–	0	0	N/A
004	Disable Input Interrupt	010aaa- -----0100	–	–	–	0	0	N/A
005	Enable Input Interrupt	010aaa- -----0101	Call Response	–	–	0	0	Input word waiting/disconnect
006	Receive N Words from Device	010aaa- -----0110	Call Response	–	00LLLL	0	0	Zero length/disconnect
007	Transfer N Words to Channel	010aaa- ------0111	–	–	00LLLL	0	0	N/A
010	Transfer N Words from Channel	010aaa- -----1000	–	–	00LLLL	0	0	N/A
011	Send N Words to Device & Disconnect	010aaa- -----1001	Call Response	–	00LLLL	0	0	Zero length
012	Send N Words to Device	010aaa- -----1010	Call Response	–	00LLLL	0	0	Zero length
013	Send Disconnect	010aaa- -----1011	–	–	–	0	0	N/A (aborts output transfer)
014	Read Output Status	010aaa- -----1100	–	–	–	0	0	N/A (clears output interrupt)

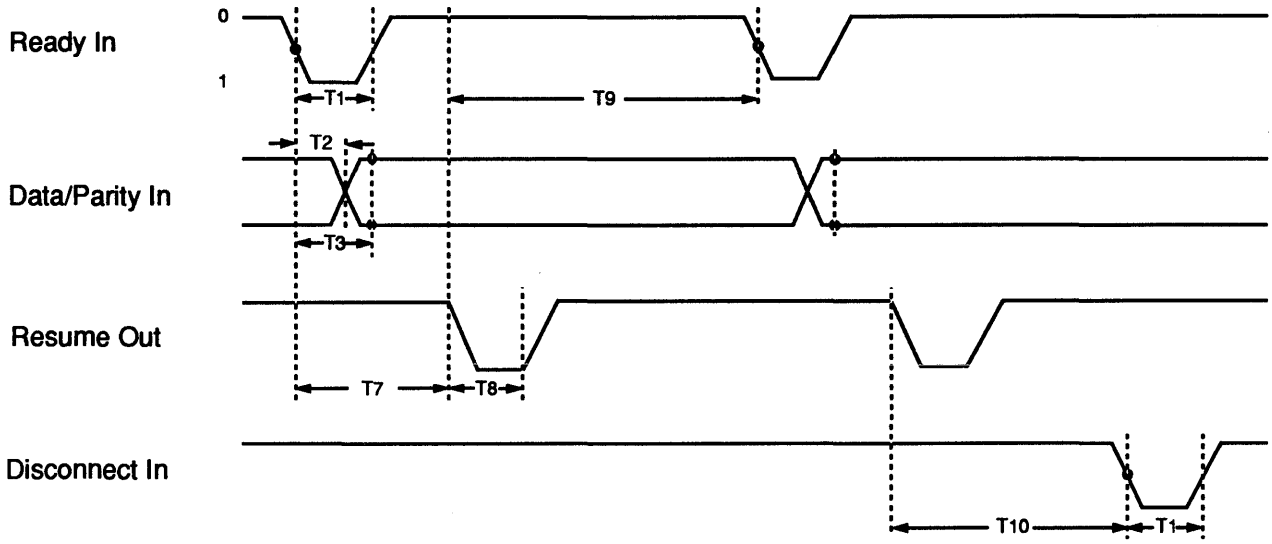
aaa Three-bit node address

LLLL **Transfer length (maximum 01000)**

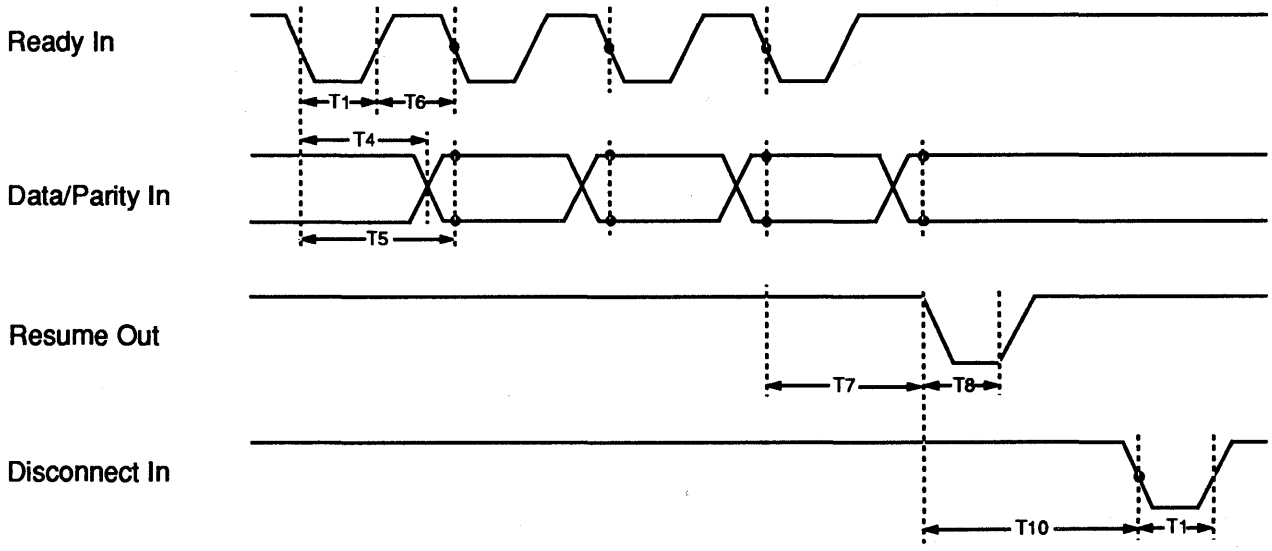
Status-high/Status-low Bit Assignments

31-28	**Input Parity Error Groups**	**15-14**	**Input Buffer Parity Errors**	**11**	**Input Disconnected**
27-26	**Output Buffer Parity Errors**	**13**	**Output Resume Error**	**10**	**Input Word Waiting**
25-16	**Output Residual Length**	**12**	**Early Input Disconnect**	**09-00**	**Input Residual Length)**

Input Channel External Timing - Low-Speed Mode



Input Channel External Timing - High-Speed Mode



T1 = Ready and Disconnect pulse width is 50 nsec (±10 nsec)

T2 = Leading edge of Ready to Input Data transition less than 30 nsec - Low-Speed
T3 = Input Data and Parity is sampled 50 nsec after leading edge of Ready - Low-Speed

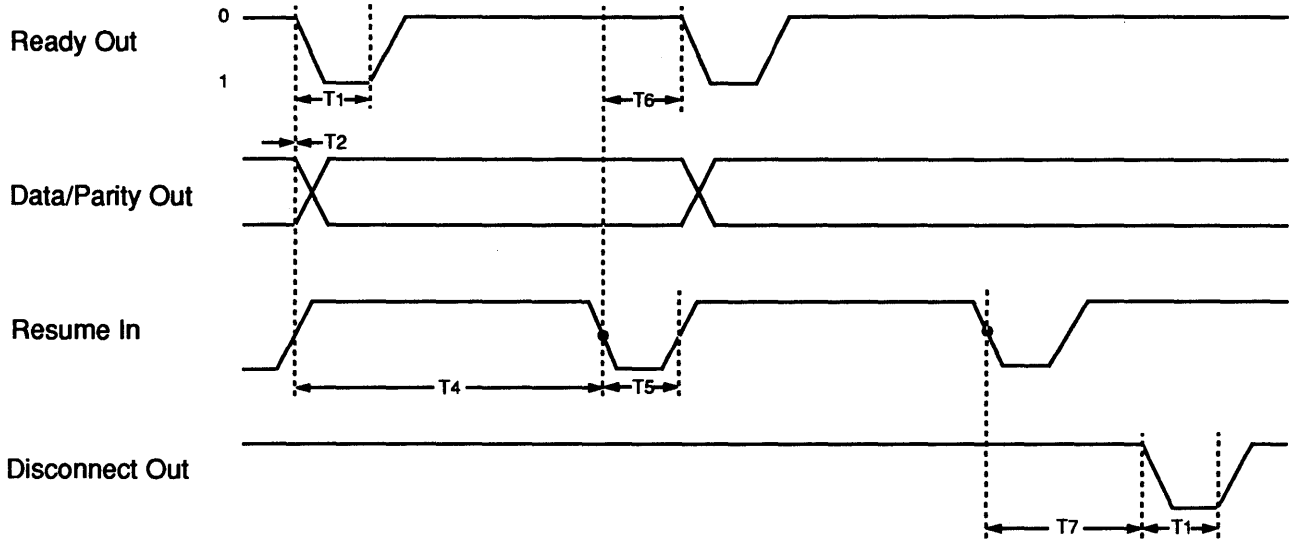
T4 = Leading edge of Ready to Input Data transition less than 80 nsec - High-Speed
T5 = Input Data and Parity is sampled 100 nsec after leading edge of Ready - High-Speed
T6 = Time between Ready pulses is 50 nsec (±10 nsec) - High-Speed

T7 = Time between Ready and Resume is 100 nsec
T8 = Resume pulse width is 50 nsec

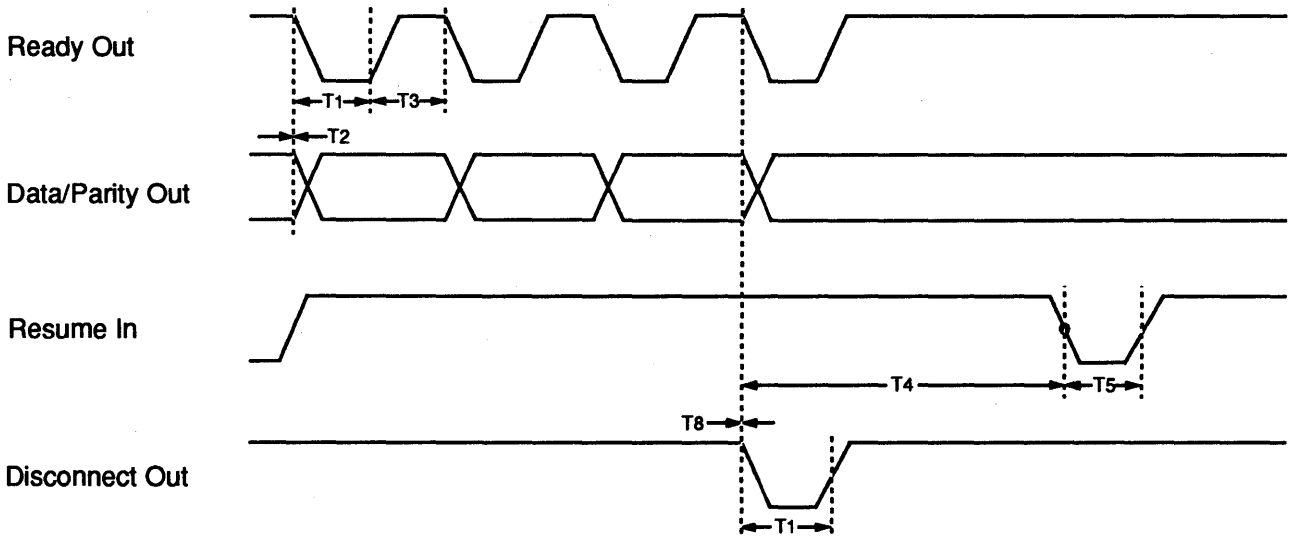
T9 = Time between Resume and next Ready is determined by round-trip cable length and device response time

T10 = Normal Disconnect is received after device receives final Resume
Time is determined by round-trip cable length and device response time

Output Channel External Timing - Low-Speed Mode



Output Channel External Timing - High-Speed Mode



T1 = Ready and Disconnect pulse width is 50 nsec

T2 = Output Data and Parity transition with leading edge of Ready pulse

T3 = Time between Ready pulses is 50 nsec - High-Speed

T4 = Time between Ready and Resume is determined by round-trip cable length and device response time

T5 = Resume pulse width is 50 nsec (±10 nsec)

T6 = Next Ready is sent 50 nsec after leading edge of Resume is detected

T7 = Normal Disconnect pulse is sent 100 nsec after final Resume is detected - Low-Speed

T8 = Normal Disconnect pulse is sent coincident with final Ready pulse - High-Speed

Appendix

C Disk Controller Interface

The disk controller interface provides for connection of DD-49 and DD-40 disk drives. The details of the interface are presented.

C.1 Disk Controller Interface

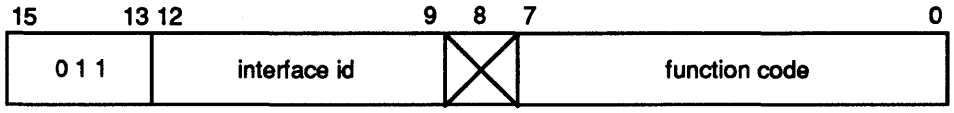
This controller contains the control necessary to interface between a DD-49 or DS-40 disk subsystem and a foreground channel loop. The controller communicates with the disk drive subsystem using two 24 pair cables, each with a 16 bit data path. Control signals to the drive include a 12.5 megahertz Write Clock, Function Ready, and a 4 bit Drive Function Code. Control signals from the drive include a 12.5 megahertz Read Clock, Status/Data Ready, Error, Done, and Drive Ready. The Index/Sector Mark signal is ignored. Data is buffered between the disk subsystem and the channel loop using two alternating read/write buffers.

C.2 Disk Controller Function Word Translation

The disk controller interface recognizes functions by translating the high order seven bits of the function word as the word passes the channel node.

The significance of the function word bit values are as follows:

Disk Controller Interface (DD40 or DD49)



Bit 15	zero value
Bit 14	one value
Bit 13	one value
Bit 12	controller id bit 3
Bit 11	controller id bit 2
Bit 10	controller id bit 1
Bit 09	controller id bit 0
Bit 08	unused
Bit 07	function code bit 7
Bit 06	function code bit 6
Bit 05	function code bit 5
Bit 04	function code bit 4
Bit 03	function code bit 3
Bit 02	function code bit 2
Bit 01	function code bit 1
Bit 00	function code bit 0

C.3 Controller General Description

The DS-40/DD-49 controller communicates with the foreground channel loop, two buffers, and the disk drive interface. It contains a controller status register, a call response register, a bus-in and drive status register, and a bus-out register. All internal data paths are 16 bits wide. The controller status register indicates continue write modes, invalid drive status, buffer parity errors, lost function, drive signal status.

The channel interface circuitry contains the call response register and relay paths for channel loop data and control. Arriving channel loop data is made available for function parameters, drive function translation, drive parameters, and buffer write data. Buffer readout data, status register contents, and blank response data are merged for transmission to the channel loop.

Each of the two buffers has a full capacity of 16384 parcels of which only the first 2048 are used. A read buffer pointer and a write buffer pointer indicate which buffer is to receive and supply data. A buffer pointer is toggled when the data transfer from the indexed buffer is complete. Independent pointers and data paths enable a data transfer to or from one buffer while data is concurrently transferred from or to the other buffer. The ping-pong buffers allow continuous disk read and write sequences that can avoid costly missed disk revolutions. No transfer counters were required in the controller since buffer transfer sizes are either 16 or 2048 parcels, and each buffer provides its current address value.

Many of the functions from the foreground processor contain parameters used in the disk functions. A continue read or write function may be pending requiring these parameters be held until the function is sent to the drive. Function parameters and write data are merged into the bus-out register for transmission over Cable A. The 8 bit controller functions are translated to 4 bit drive function codes and held for transmission. The module clock is divided by 40 to produce a nominal 12.5 megahertz Write Clock signal to the drive. The Function Ready pulse is synchronized to the positive edge of this clock and lasts for one Write Clock period. The drive busy flag is set when a Function Ready pulse is pending that prevents acceptance of concurrent drive functions. A two nanosecond pulse is provided at the negative edge of Write Clock to initiate a buffer read sequence for each parcel sent during data transfers to the disk drive.

The Read Clock signal from the disk drive is sampled for three consecutive ones followed by a sampled zero to detect its negative edge. The Status/Data Ready, Error, and Done signals are sampled when the negative edge is detected. The bus-in and drive status registers are entered with Cable B data under certain combinations of these drive signals. Captured bus-in data is made available to each buffer for read function data transfers. The drive busy flag is reset when a Done pulse is received.

C.4 Controller Status Register

This status register contains 14 bits that provide status of the controller and drive signals. It is placed in the second parcel of response data for the Read

Controller Status function (06x). A Read Controller Status function will reset some of these bits. The significance of each bit is given below.

Bit 15	Unused
Bit 14	Unused
Bit 13	Last Continue Write Done*
Bit 12	Wait Interrupt Last Continue Write*
Bit 11	Read Status Invalid *
Bit 10	Buffer B Read Parity Error*
Bit 09	Buffer A Read Parity Error*
Bit 08	Lost Function*
Bit 07	Bus-in Parity
Bit 06	Status Parity
Bit 05	Bus-in Parity Error*
Bit 04	Status Parity Error*
Bit 03	Drive Error*
Bit 02	Drive Busy/Invalid Drive Command*
Bit 01	Drive Status Available*
Bit 00	Drive Ready

* These bits are reset when this register is read

C.4.1 Bit 13 - Last Continue Write Done

This bit indicates that the drive status is valid for the last write. The drive status is invalid if this bit and bit 12 are set. This bit is reset when the status register is read.

C.4.2 Bit 12 - Wait Interrupt Last Continue Write

This bit indicates the last continue write operation is not complete. Present drive status is for the previous write. The drive status is invalid if this bit and bit 13 are set. This bit is reset when the status register is read.

C.4.3 Bit 11 - Read Status Invalid

This bit indicates that the drive status was overwritten before the drive status was read. It is reset when the status register is read.

C.4.4 Bit 10 - Buffer B Read Parity Error

This bit indicates a parity error was detected in Buffer B during a buffer read operation. This bit is reset when the status register is read.

C.4.5 Bit 09 - Buffer A Read Parity Error

This bit indicates a parity error was detected in Buffer A during a buffer read operation. It is reset when the status register is read.

C.4.6 Bit 08 - Lost Function

This bit indicates that a channel function could not be processed and was ignored. The Read Status function is always accepted. Disk Read and Write functions will set this bit if a read or write function is pending. All other functions will set this bit if the drive busy flag is set. This bit is reset when the status register is read.

C.4.7 Bit 07 - Bus-in Parity

This bit is a sample of the Bus-in Parity signal. It is sampled at the negative edge of Read Clock. This bit is not affected by the Read Status function.

C.4.8 Bit 06 - Status Parity

This bit is a sample of the Status Parity signal. It is sampled at the negative edge of Read Clock. This bit is not affected by the Read Status function.

C.4.9 Bit 05 - Bus-in Parity Error

This bit indicates that a parity error was detected on the bus-in data. This bit is reset when the status register is read.

C.4.10 Bit 04 - Status Parity Error

This bit is set when the sampled Status/Data Ready, Error, Done, and Status Parity signals have invalid parity while Drive Ready is asserted. This bit is reset when the status register is read.

C.4.11 Bit 03 - Drive Error

This bit is set when the Error signal is sampled and is asserted, indicating that the drive detected an error during the previous operation. This bit is reset when the status register is read.

C.4.12 Bit 02 - Drive Busy/Invalid Drive Command

This bit is set when the Status/Data Ready, Error, and Done signals are sampled and all asserted, indicating that the drive is busy (connected to the other port), the drive received and invalid command, or a drive function failed. Drive status is available when this bit is set. This bit is reset when the status register is read.

C.4.13 Bit 01 - Drive Status Available

This bit indicates that the drive status register contents are valid. It is set when Status/Data Ready and Done are sampled asserted, and Error was not asserted. This bit is reset when the status register is read.

C.4.14 Bit 00 - Drive Ready

This bit is simply a copy of the Drive Ready signal. It indicates that the drive is ready to accept commands. This bit is not affected by the Read Status function.

C.5 Drive Status Register

This status register contains 16 bits that are captured from the disk subsystem. The value on Cable B from the drive is sampled when the Done pulse is received. The status register's contents are placed in the first parcel of response data for the Read Controller Status function (function 06x). The Read Controller Status function will clear all bits.

The contents of the Drive Status register depends on the last drive function executed. If an Error pulse was detected at the completion of any disk function, the register's contents are invalid. Successful completion of functions Diagnostics Select (07x), Select Cylinder (100), Select Head (101), Read (20x,21x), Write (30x,31x), and Echo Parameter Word (37x) enter the bus-out value at the start of the function into this register when done. The Select Status function (04x) captures the selected drive status register value. The remaining disk functions return the drive's general status register value into the controller's Drive Status register. See General Drive Status function 05x for a description of the DD-49 and DS-40 general status registers.

C.6 Call Response Register

This 16 bit register holds the call response data given in the second parcel of disk functions. Its contents replace the second parcel following a channel call pulse if an interrupt request is enabled. The call response data is held until it is replaced with another value from a new function request.

C.7 Cable A to Drive Signal Descriptions

C.7.1 Write Clock

This signal is generated for synchronization of commands and data to the drive. Function code and data transitions occur on the low-to-high transition of this clock. A high-to-low transition defines the center of a bus-out cycle. Period of this signal at the drive is 75 nanoseconds (+10.74 nsec, -2.1 nsec). The controller produces a nominal 12.5 megahertz Write Clock (80 nanosecond period). Set-up and hold time of the data with respect to the falling edge of this clock is 20 nanoseconds.

C.7.2 Function/Data Ready

This signal is asserted during the bus-out cycles which carry a valid function code on the function code lines, or valid write data on the bus-out lines. The signal is pulsed for a single period of Write Clock.

C.7.3 Function Code - 4 Bits

These four bits carry the function code to be performed by the drive. Function codes are recognized during bus-out cycles in which Function/Data Ready is asserted and function code parity is valid.

C.7.4 Function Parity

This signal carries odd parity of the four function code bits. Parity is checked only during bus-out cycles in which Function/Data Ready is asserted.

C.7.5 Bus-Out - 16 Bits

These 16 bits carry function parameters and write data to the drive. Data on these lines are valid only when Function/Data Ready is asserted.

C.7.6 Bus-Out Parity

This signal carries odd parity of the 16 bus-out data bits. Parity is checked only during bus-out cycles in which Function/Data Ready is asserted.

C.8 Cable B from Drive Signal Descriptions

C.8.1 Read Clock

This signal is generated by the drive for synchronization of status and read data from the drive. Status and data transitions occur on the low-to-high transition of this clock. A high-to-low transition defines the center of a bus-in cycle. Period of this signal from the drive is 80 nanoseconds (0.5 nanoseconds). Set-up and hold time of status and data with respect to the falling edge of the clock is 20 nanoseconds. Read Clock will be transmitted from the drive regardless of whether or not the port is selected. The signal is inhibited when the port is disabled.

C.8.2 Status/Data Ready

This signal is asserted when the drive is presenting status or read data on the bus-in data lines. It is also asserted when the Select functions return a busy response when the drive is reserved to the other port. The signal is pulsed for a single period of Read Clock.

C.8.3 Error

This signal is asserted in conjunction with the Done signal to indicate that at least one error condition occurred during the current function. It is pulsed for a single period of Read Clock.

C.8.4 Done

This signal indicates the completion of a drive command. It is pulsed for a single period of Read Clock.

C.8.5 Ready

This signal indicates that the drive is ready to accept commands. Ready does not imply that the spindle is sequenced up. Ready does imply that the drive port is enabled (whether selected or not), or that the drive port is selected (the Port Enable Switch is depressed). If the General Status Word indicates that a

spindle sequence is in progress, then only functions Select, General Status, Release, and Release Opposite and Select can be executed by the drive without getting an error completion.

C.8.6 Index/Sector Mark

This signal carries encoded index and sector mark information. It is asserted for one Read Clock cycle to indicate a sector mark. It is asserted for two consecutive Read Clock cycles to indicate index mark. This signal will be transmitted from the drive regardless of whether or not the port is selected. The signal is inhibited when the port is disabled.

The index pulse is generated at the beginning of sector 0. Sector marks are generated at the end of each physical sector on the track.

C.8.7 Status Parity

This signal carries odd parity of the four status signals Status/Data Ready, Error, Done, and Ready. Parity is valid only during bus-in cycles in which Ready is asserted.

C.8.8 Bus-In - 16 Bits

These 16 bits carry status register information and read data from the drive. Data on these lines are valid only when Status/Data Ready or Done is asserted.

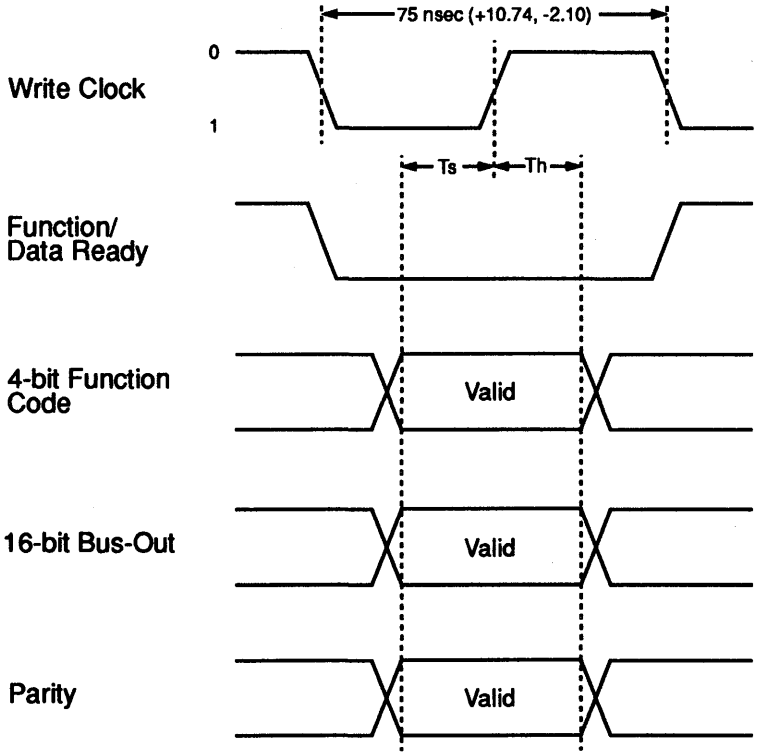
C.8.9 Bus-In Parity

This signal carries odd parity of the 16 bus-in data bits. Parity is valid only during bus-in cycles in which Ready is asserted.

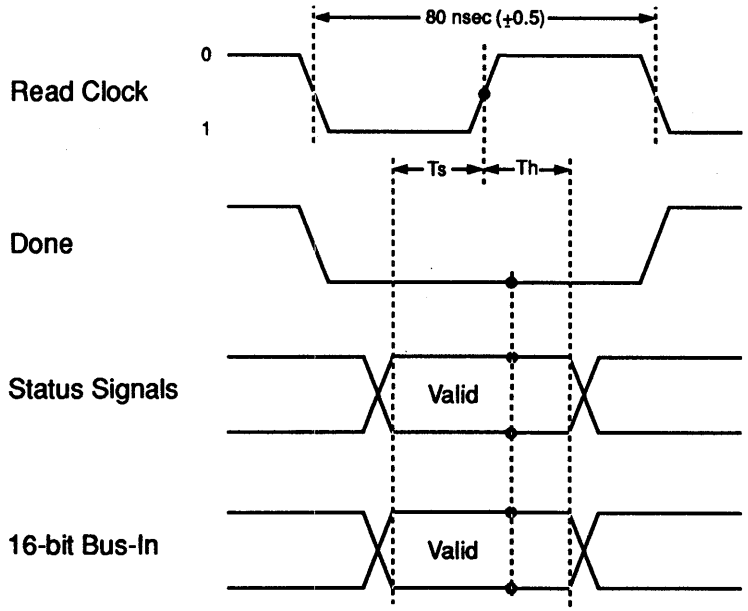
C.9 Cable Signal Timing

The diagrams that follow illustrate timing of the cable signals as they appear at the controller. Signal timing relative to the appropriate clock signal for a single clock period appears on page 304. In the Cable B timing diagram, circles indicate sample points on the input signals. The sample point on the falling edge of Read Clock shows that three consecutive ones followed by a zero was sampled for edge detection. The edge-detection samples are taken every 8 nanoseconds. Therefore, the time between Read Clock edge detection and signal capture will be 0 to 8 nanoseconds.

Cable A Timing
(Signals to Drive)



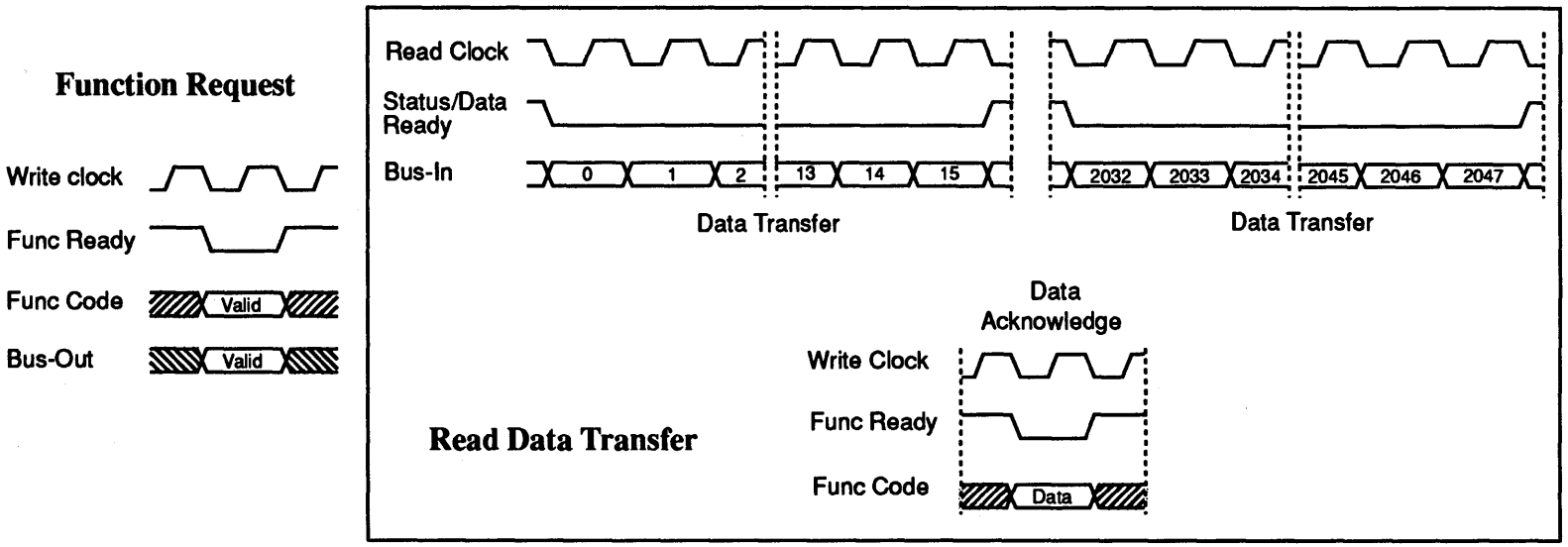
Cable B Timing
(Signals from Drive)

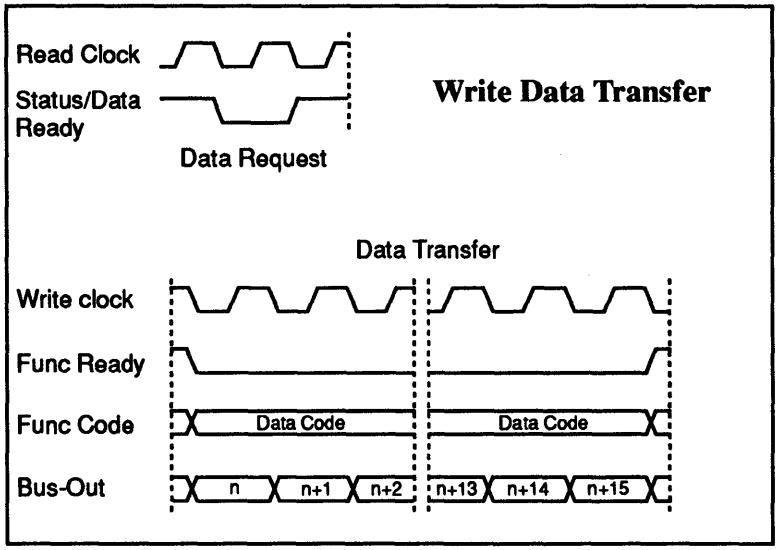


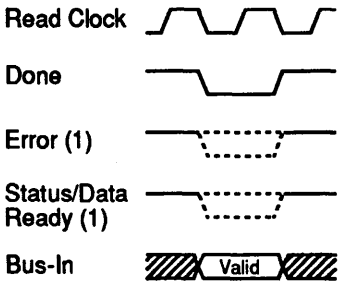
Notes:

1. Write Clock is 50% duty cycle. Controller provides 80 nsec period.
2. Read Clock period is 50% duty cycle.
3. Maximum 4.5 nsec rise and fall times.
4. Setup time Ts = 20 nsec.
5. Hold time Th = 20 nsec.

Drive Function Timing Sequence







Notes:

1. Error and Status/Data Ready are asserted only under certain conditions.

CRAY-3 DS-40/DD-49 Disk Controller Functions

Fcn	Function Name	A Register		B Register		Function Response		Respond to call after:
		Parcel-0	Parcel-1	Parcel-2	Parcel-3	Parcel-0	Parcel-1	
000	Release	011aaaa -00000000	Call Response	–	002524	0	0	Release function done.
001	Release Opposite and Select	011aaaa -00000001	Call Response	–	002524	0	0	Release opposite and select done.
01x	Select	011aaaa -00001---	Call Response	–	002524	0	0	Reserve attempt done.
020	Clear Faults	011aaaa -00010000	Call Response	–	–	0	0	Clear to DSU done.
021	Reset	011aaaa -00010001	Call Response	–	–	0	0	Reset to DSU done.
03x	Return-To-Zero Cylinder	011aaaa -00011---	Call Response	–	–	0	0	Cylinder zero found, DSU done.
04x	Select Status	011aaaa -00100---	Call Response	–	Number	0	0	Status received from DSU.
05x	General Drive Status	011aaaa -00101---	Call Response	–	–	0	0	General status received from DSU.
06x	Read Controller Status	011aaaa -00110---	–	–	–	Drive Sts	Contrllr Sts	N/A
07x	Diagnostic Select	011aaaa -00111---	Call Response	–	Mode	0	0	Diagnostics function, status.
100	Select Cylinder	011aaaa -01000000	Call Response	–	Offset/Cyl	0	0	Seek done, status returned.
101	Select Head Group	011aaaa -01000001	Call Response	–	Head	0	0	Head Select done.
200	Read Data Sector	011aaaa -10000000	Call Response	Nxt Hd/Sctr	Cont/Sector	0	0	2048 parcels from drive.
201	Read Sector ID	011aaaa -10000001	Call Response	–	Sector	0	0	16 parcels from drive.
202	Read Absolute Data Sector	011aaaa -10000010	Call Response	–	Sector	0	0	2048 parcels from drive.
203	Read Buffer	011aaaa -10000011	Call Response	–	–	0	0	16 parcels from drive.
204	Read ECC Block	011aaaa -10000100	Call Response	–	–	0	0	16 parcels from drive.
205	Read Error Correction Vectors	011aaaa -10000101	Call Response	–	–	0	0	16 parcels from drive.
206	Read Track Header	011aaaa -10000110	Call Response	–	Sector	0	0	16 parcels from drive.
210	Read Data Sector, No Read Ahead	011aaaa -10001000	Call Response	–	Sector	0	0	2048 parcels from drive.
212	Read Sector from Track Buffer	011aaaa -10001010	Call Response	–	Sector	0	0	2048 parcels from drive.
25x	Read Buffer Data to Channel	011aaaa -10101---	–	–	–	Delayed 0	Delayed 0	N/A
300	Write Data Sector	011aaaa -11000000	Call Response	Head	Cont/Sector	0	0	2048 parcels from drive.
301	Write Sector ID	011aaaa -11000001	Call Response	–	Sector	0	0	16 parcels from drive.
302	Write Defective Sector ID	011aaaa -11000010	Call Response	–	Sector	0	0	16 parcels from drive.
303	Write Buffer	011aaaa -11000011	Call Response	–	–	0	0	16 parcels from drive.
304	Write Sector with Zero ECC Block	011aaaa -11000100	Call Response	–	Sector	0	0	2048 parcels from drive.
306	Write Track Header	011aaaa -11000110	Call Response	–	–	0	0	16 parcels from drive.
310	Write Data Sector, No Write Ahead	011aaaa -11001000	Call Response	–	Sector	0	0	2048 parcels from drive.
314	Write Data Sector to Track Buffer	011aaaa -11001100	Call Response	–	–	0	0	2048 parcels from drive.
36x	Buffer Echo	011aaaa -11110---	–	–	Buffer	0	0	Call pulse clears buffer echo mode
37x	Echo Parameter Word	011aaaa -11111---	Call Response	–	Echo Data	0	0	Echo word received.

Note:

aaaa 4-bit node address
– Ignored bit/parcel

The general sequence of signals involved in function request, data transfer, and function completion with the drive appears on page 305. Most of the drive functions, such as Echo Parameter Word, will only involve the Function Request and Function Done cycles.

C.10 Controller Functions

The function field is taken from the low order eight bits of the channel function parcel. The function field is then translated for the specific action requested by the foreground processor. These functions are listed below:

C.10.0.1 DS-40/DD-49 Controller Functions

000	Release
001	Release Opposite and Select
01x	Select
020	Clear Faults
021	Reset
03x	Return-To-Zero Cylinder
04x	Select Status
05x	General Drive Status
06x	Read Controller Status
07x	Diagnostic Select
100	Select Cylinder
101	Select Head Group
200	Read Data Sector
201	Read Sector ID
202	Read Absolute Data Sector
203	Read Buffer
204	Read ECC Block
205	Read Error Correction Vectors
25x	Read Buffer Data to Channel
300	Write Data Sector
301	Write Sector ID

302	Write Defective Sector ID
303	Write Buffer
304	Write Data Sector with Zero ECC Block
36x	Buffer Echo
37x	Echo Parameter Word

C.10.0.2 DS-40 Functions

206	Read Track Header
210	Read Data Sector, No Read Ahead
212	Read Data Sector from Track Buffer
306	Write Track Header
310	Write Data Sector, No Write Ahead
314	Write Data Sector to Track Buffer

C.11 Function Descriptions

The following descriptions discuss each function's action in the controller and in the drive. Since this controller may be used for a DS-40 subsystem as well as a DD-49 drive, some functions contain separate descriptions for each drive subsystem type. These sections have the headings DD-49 and DS-40.

Functions 06x, 25x, and 36x are not specifically disk functions since they only initiate internal controller activity. All other functions are disk functions that send the following 4 bit function codes to the drive:

Function Code	4 bit Code	Function Name
000	1111	Release
001	1110	Release Opposite and Select
01x	0001	Select
020	1100	Clear Faults
021	1011	Reset
03x	1101	Return To Zero
04x	0111	Select Status
05x	1000	General Drive Status

Function Code	4 bit Code	Function Name
07x	1001	Diagnostic Select
100	0101	Cylinder Select
101	0100	Head Select
20x,21x	0010	read data functions
30x,31x	0011	write data functions
37x	0000	Echo Parameter Word

At the beginning of all disk functions, the controller captures the call response from parcel one and captures function parcels two and three into two parameter registers. Zero response data is immediately sent to the foreground processor. The controller then sends the appropriate 4 bit function code and function parcel three over bus-out to the drive. If the drive is busy with a read command when a read function is received from the foreground processor, the read function is queued in the controller and sent when the current read command is done.

After the initial function sequence, the read and write functions will transfer data when the drive is not busy. A further description of controller activity for the read and write functions is given in their function descriptions. The remaining disk functions simply wait for the Done pulse from the drive.

When a drive command completes, a Done pulse is received from the drive that terminates the disk controller function, captures bus-in into the Drive Status register, and enables the controller to respond to a channel call. The Drive Status register is only valid upon successful completion (no error).

C.11.1 Function 000 - Release

This command breaks the drive reservation established in the Select or Release Opposite and Select commands. The Release function is effective only when issued by the active port, and makes the drive available for reservation by either port. This command re-establishes the default drive conditions. Upon successful completion, the general status register value is available in the controller Drive Status register.

DD-49:

This function requires a value of 002524_8 in function parcel three.

DS-40:

Function parcel three bits 2-0 contain the unit number. A unit number of 0 indicates the primary drive, 1 indicates the shadow drive, and 7 means port only.

C.11.2 Function 001 - Release Opposite and Select

This command breaks the current drive reservation regardless of port, and the port issuing the command is selected. Issuing this command from the selected port does not cause an error. The command is only recognized between command executions on the port that is to be released, allowing a more orderly release. However, this command should be used with extreme caution. Upon successful completion, the general status register value is available in the controller Drive Status register.

DD-49:

This function requires a value of 002524_8 in function parcel three.

DS-40:

Drive Status register bits 2-0 contain the unit number. A unit number of 0 indicates the primary drive, 1 indicates the shadow drive, and 7 means port only.

This function can select a diagnostic loopback. The loopback mode is used with the Echo Parameter Word function. Loopback mode forces the parameters from the controller to loop through the DS-40 and back. Loopback mode is selected by setting bits 2 through 0 in function parcel three to all ones (7).

C.11.3 Function 01x - Select

This function attempts to logically connect this channel to the drive, locking out the other port to the drive. This function must be issued successfully before most other drive functions are allowed. Upon successful completion, the general status register value is available in the controller Drive Status register.

DD-49:

This function requires a value of 002524_8 in function parcel three.

DS-40:

Drive Status register bits 2-0 contain the unit number. A unit number of 0 indicates the primary drive, 1 indicates the shadow drive, and 7 means port only.

C.11.4 Function 020 - Clear Faults

The Controller Status Register and the status register in the drive is cleared. The drive remains ready throughout the function. A Return To Zero Cylinder is not performed. Therefore, any seek error conditions are not reset. Upon successful completion, the general status register value is available in the controller Drive Status register.

C.11.5 Function 021 - Reset

The reset command causes the drive to go through a reset sequence if the opposite port is not selected. Port selection is retained if the issuing port is selected at the time of the reset command. Upon successful completion, the general status register value is available in the controller Drive Status register. This function does not clear the Controller Status register.

DD-49:

The following reset sequence is performed in the DD-49:

1. Hardware reset both MPU card processors
2. Reset all fault conditions and status
3. Reset all software implemented status
4. Execute a subset of the power-on diagnostics
5. Perform a return-to-zero (RTZ) command
6. Clear the 16 write buffer data words
7. Reestablish default drive conditions and clear maintenance mode bit

The reset command does not include the servo calibration procedure and the 30-minute warm-up period is not restarted. Default drive conditions are reestablished (data cylinders are write enabled and FE cylinders are write protected).

DS-40:

The following reset sequence is performed in the DS-40:

1. Reset all fault conditions and status
2. Perform a return-to-zero (RTZ) command

C.11.6 Function 03x - Return-to-Zero Cylinder

This command clears all seek-related faults and offsets, and repositions the read/write heads to Cylinder 0. Head selection is unaffected by this command.

Upon successful completion, the general status register value is available in the controller Drive Status register.

C.11.7 Function 04x - Select Status

This function selects a status register in the drive specified in function parcel three. Upon successful completion, a Read Controller Status function provides the selected status in the Drive Status register. Refer to the drive operator's manual for format of all registers.

DD-49:

The DD-49 uses the lowest 5 bits of bus-out (function parcel three) to request one of 24 status registers from the drive. These status registers are listed below:

0	Version/Revision
1	Echo
2	Cylinder/Offset
3	Current Head
4	Current Sector
5	Last Non-status Command
6	Last Non-status Command Option
7	Drive Sense 1
8	Extended Status 1
9	Extended Status 2
10	DSU Controller Fault Codes
11	Fault Code Parameters
12	Supervisor Fault Codes
13	Maintenance Status
14	Warm-up Timer
15	Velocity Scale Factors
16	Extended Status 3
17	Extended Status 4
18	Servo Sense 1
19	Servo Sense 2
20	Sector "N" Channel Status
21	Sector "N+1" Channel Status

22	Sector "N+2" Channel Status
23	Diagnostic Status Word

DS-40:

The status selection bit assignments encoded in function parcel three that is sent to the drive are as follows:

Bits 15-12	Unused
Bit 11	Bit 4 select drive status
Bit 10	Bit 3 select drive status
Bit 09	Bit 2 select drive status
Bit 08	Bit 1 select drive status
Bit 07	Bit 0 select drive status
Bit 06	Status selection bit (0 = DS-40 controller, 1 = DSU)
Bit 05	Drive spindle select bit 1
Bit 04	Drive spindle select bit 0
Bit 03	Sector number select bit 3
Bit 02	Sector number select bit 2
Bit 01	Sector number select bit 1
Bit 00	Sector number select bit 0

C.11.8 Function 05x - General Drive Status

This function reads the drive's general status register into the controller Drive Status register. The following describes each bit in the DD-49 and DS-40 general status registers.

DD-49 General Status Register

Bit 15	In Maintenance Mode
Bit 14	Drive Fault
Bit 13	Sequence Operation in Progress
Bit 12	Invalid Command
Bit 11	Function Lost
Bit 10	Sync Timeout
Bit 09	ID Not Found

Bit 08	ECC Error Channel B2
Bit 07	ECC Error Channel B1
Bit 06	ECC Error Channel A2
Bit 05	ECC Error Channel A1
Bit 04	Invalid Option or Argument
Bit 03	Seek Error
Bit 02	Data Underflow/Overflow
Bit 01	Bus-Out Parity Error
Bit 00	Function Parity Error

- Bit 15: Indicates the drive is currently in the Maintenance mode of operation. At least one of the Write Enable/Protect Control bits specified by the select diagnostic function is nonzero. The drive is not the default condition of the write enable user cylinders and the write protected FE cylinders.
- Bit 14: Indicates an unrecoverable error in the drive logic was detected.
- Bit 13: Indicates the drive spindle sequence up/down operation is in progress because of a change of the drive's front panel Run switch. Only Select, Release, Release Opposite and Select, and General Status functions will run successfully.
- Bit 12: Indicates the drive detected a data function (0110) outside the context of read/write or detected and unused function (1010).
- Bit 11: Indicates the drive detected an unexpected Function Ready signal.
- Bit 10: Indicates the sync bytes for each drive channel were not found within the allowed window for the field.
- Bit 09: Indicates the ID was not found on the specified physical sector, if that sector was formatted as defective, the following 2 sectors (slipped sectors).
- Bit 08: Indicates an error on drive channel B2 during the last read operation.
- Bit 07: Indicates an error on drive channel B1 during the last read operation.
- Bit 06: Indicates an error on drive A2 during the last read position.
- Bit 05: Indicates an error on drive channel A1 during the last read operation.
- Bit 04: Indicates an invalid option or argument was received.
- Bit 03: Indicates a drive fault was encountered during a seek operation.

- Bit 02: Indicates an error occurs if the controller does not pass data to the drive fast enough on a write (underflow), or if the controller does not take data from the drive fast enough on a read (overflow).
- Bit 01: Indicates an error was detected during the handling of data.
- Bit 00: Indicates an error was detected during the reception of the function codes from the controller.

DS-40 General Status Register

Bit 15	Defect parity error (sector status of last error)
Bit 14	Sync time-out (sector of last error)
Bit 13	ID not found (sector of last error)
Bit 12	ECC error (sector status of last error)
Bit 11	Drive fault on any drive in DS-40
Bit 10	Seek error on any drive in DS-40
Bit 09	On cylinder on all drives in DS-40
Bit 08	Unit ready on all drives in DS-40
Bit 07	Buffer error
Bit 06	Channel error
Bit 05	Drive number of most recent error bit 1
Bit 04	Drive number of most recent error bit 0
Bit 03	Sector number of most recent error bit 3
Bit 02	Sector number of most recent error bit 2
Bit 01	Sector number of most recent error bit 1
Bit 00	Sector number of most recent error bit 0

- Bit 15: Indicates a parity error.
- Bit 14: Indicates the sync for each drive channel was not found within the allowed window for the field.
- Bit 13: Indicates the ID was not found on the specified sector.
- Bit 12: Indicates an error on the drive channel during the last read operation.
- Bit 11: Indicates an unrecoverable error in the drive logic.
- Bit 10: Indicates a drive fault was encountered during a seek operation.
- Bit 09: Indicates an on cylinder status for all drives in the DS-40 subsystem.

- Bit 08: Indicates a unit ready condition for all drives in the DS-40 subsystem.
- Bit 07: Indicates a track buffer error.
- Bit 06: Indicates a bus-out channel error (function parity error, bus parity error, or command error).
- Bits 05,04: Indicates the drive number of the most recent error.
- Bits 03-00: Indicates the sector number of the most recent error.

C.11.9 Function 06x - Read Controller Status

The controller ignores the remaining function parcels and responds immediately with function response data. The first response parcel contains the Drive Status register and the second parcel contains the Controller Status register. All bits except Bus-in Parity, Status Parity, and Drive Ready in the Controller Status register are cleared when this function is done. The Drive Status register is not cleared.

C.11.10 Function 07x - Diagnostic Select

This function performs a diagnostics procedure in the disk subsystem. Upon successful completion, the Drive Status register will contain the value of function parcel three. If a Parity error is requested, the Controller Status Register will show. Refer to the drive's operation manual for bus-out codes.

DD-49:

Function parcel three specifies the diagnostics function requested as follows:

Function Parcel 3	Description
000001	Forces a parity error on bus-in. The bus-in parity bit of status register (word 1) should be set upon completion.
000002	Forces a parity error on status. The status parity bit of status register (word 1) should be set upon completion.
000004	Write enable FE1 (Cylinder 887) allows FE cylinder 1 to be written. FE1 contains the flaw maps. This function sets the Maintenance Mode bit in the general drive status register.
000010	Write enable FE2 (Cylinder 889) allows FE Cylinder 2 to be written. FE2 is a diagnostic scratch cylinder. This function sets the Maintenance Mode bit in the General Drive Status register.

Function Parcel 3	Description
000020	Write protect data Cylinder 0 through 885 allows this data cylinders to be write protected. This function sets the Maintenance Mode bit in the General Drive Status register.
004001	Requests spindle sequence up
002002	Confirms spindle sequence up
004002	Requests spindle sequence down
002004	Confirms spindle sequence down
004003	Performs servo calibration
004004	Test single bit error detection and correction (EDC) for the DSU controller's memory
004005	Test multiple bit error detection (EDC) for the DSU controller's memory

The write enable-write protect diagnostic modes (the Maintenance Mode bit set in the General Drive Status register) remain in effect until one of the following occurs:

- A diagnostic select function is issued that does not request a write enable/write protect diagnostic mode
- A select function is issued
- A release function is issued
- A reset function is issued
- The drive is power cycled

DS-40:

The diagnostic functions selected by function parcel three are:

- Bit 0: Forces a parity error on bus-in. The bus-in parity bit of status register (word 1) should be set upon completion.
- Bit 1: Forces a parity error on status. The status parity bit of status register (word 1) should be set upon completion.

C.11.11 Function 100 - Select Cylinder

This function positions the read/write heads over the selected cylinder (seek). Upon successful completion, the Drive Status register will contain the value of function parcel three. Format for this parcel is given below.

DD-49:

The cylinder select bit assignments encoded in function parcel three that is sent to the drive are as follows:

Bit 15	Offset enable B
Bit 14	Offset B (1 = toward spindle, 0 = away from spindle)
Bit 13	Offset enable A
Bit 12	Offset A
Bit 11	Zero
Bit 10	Zero
Bit 09	Cylinder bit 9
Bit 08	Cylinder bit 8
Bit 07	Cylinder bit 7
Bit 06	Cylinder bit 6
Bit 05	Cylinder bit 5
Bit 04	Cylinder bit 4
Bit 03	Cylinder bit 3
Bit 02	Cylinder bit 2
Bit 01	Cylinder bit 1
Bit 00	Cylinder bit 0

- Bits 15 through 12: Specifies the offsets used for offset operations. The drive must be on the desired cylinder to initiate an offset operation. Either or both actuators may be offset by setting the appropriate actuator offset enable. The offset direction is specified for each actuator. Only a single increment of offset is available for both the forward and reverse directions.

  An Offset Direction bit set with offset enable causes the actuator to offset toward the spindle. Conversely, if the Offset Direction bit is not set with offset enable, the actuator offsets away form the spindle. Write operation to the DD-49 are disabled with an offset enabled.

- Bits 00 through 09: Specifies the cylinder to be selected on seek operations. A seek to a different cylinder operation cannot be initiated with the offsets enabled.

DS-40:

Function parcel three format:

Bit 15	Unused
Bit 14	Unused
Bit 13	Offset direction out (1 = offset away from spindle)
Bit 12	Offset direction in (1 = offset toward spindle)
Bit 11	Cylinder bit 11
Bit 10	Cylinder bit 10
Bit 09	Cylinder bit 09
Bit 08	Cylinder bit 08
Bit 07	Cylinder bit 07
Bit 06	Cylinder bit 06
Bit 05	Cylinder bit 05
Bit 04	Cylinder bit 04
Bit 03	Cylinder bit 03
Bit 02	Cylinder bit 02
Bit 01	Cylinder bit 01
Bit 00	Cylinder bit 00

- Bits 13,12: These bits request cylinder offset access. The drive must already be on the desired cylinder to initiate an offset operation. Only an offset increment is used (75 Micro in.) in both the forward and reverse directions. Write operations to the drive are disabled when an offset is enabled.
- Bits 00 through 11: These bits specify the cylinder to be selected on seek operations. A seek to a different cylinder operation cannot be initiated with the offsets enabled.

C.11.12 Function 101 - Select Head Group

This function specifies the head group for the next read/write function. Upon successful completion, the Drive Status register will contain the value of function parcel three.

DD-49:

The 3 bit head number is located in function parcel three bits 10-08, bit 08 the LSB. All other bits (15-11 and 07-00) are unused, and must be zero.

DS-40:

The 5 bit head number is located in function parcel three bits 11-07, bit 07 the LSB. All other bits (15-12 and 06-00) are unused, and must be zero.

C.11.13 Functions 200-212 - Disk Read Functions

The following read functions request data transfer from the drive:

200	Read Data Sector, Continue Read, 2048 parcels
201	Read Sector ID, 16 parcels
202	Read Absolute Data Sector, 2048 parcels
203	Read Buffer, 16 parcels
204	Read ECC Block, 16 parcels
205	Read Error Correction Vectors, 16 parcels
206	Read Track Header, 16 parcels, DS-40 only
210	Read Data Sector, No Continue Read, 2048 parcels, DS-40 only
212	Read Data Sector from Track Buffer, 2048 parcels, DS-40 only

Function parameters were captured when the read function was received. Function parcel two contains the head number, and for a continue read, the next sector number. The head number specifies the head group that will be selected after the read operation is complete. If the continue read flag is set for function 200, the next sector value is sent to the drive after the first read is done. Function parcel two has the following bit assignments:

Bit 15	Unused
Bit 14	Unused
Bit 13	Unused
Bit 12	Unused
Bit 11	Head bit 4
Bit 10	Head bit 3 / DD-49 Head bit 2
Bit 09	Head bit 2 / DD-49 Head bit 1

Bit 08	Head bit 1 / DD-49 Head bit 0
Bit 07	Head bit 0
Bit 06	Next sector bit 6
Bit 05	Next sector bit 5
Bit 04	Next sector bit 4
Bit 03	Next sector bit 3
Bit 02	Next sector bit 2
Bit 01	Next sector bit 1
Bit 00	Next sector bit 0

- Bits 11 through 07: These bits specify the logical head group to be selected after this read operation. Bits 10-08 specify head groups 0 through 7 for a DD-49, and bit 11 must be zero. For a DS-40, bits 11-07 specify head groups 0 through 18.
- Bits 00 through 06: These bits specify the next sector number for a continue read operation. If the Continue read bit is set, the controller sends this sector number for all reads after the first request (applies to function 200 only).

Function parcel three contains the continue read flag (for function 200 only) and the first sector number. If the continue read flag is set, only the first continue read request will use the sector number in this parcel. Function parcel three has the following bit assignments:

Bit 15	Continue read flag, function 200 only
Bit 14	Unused
Bit 13	Unused
Bit 12	Unused
Bit 11	Unused
Bit 10	Unused
Bit 09	Unused
Bit 08	Unused
Bit 07	Unused
Bit 06	First sector bit 6
Bit 05	First sector bit 5
Bit 04	First sector bit 4
Bit 03	First sector bit 3

Bit 02	First sector bit 2
Bit 01	First sector bit 1
Bit 00	First sector bit 0

- Bit 15: This bit enables the controller to automatically read the next sector specified in parcel two after the first read operation is complete.
- Bits 06 through 00: The sector number specifies the logical sector to be read. The ID field for this sector should match the current cylinder, current head, and logical sector specified.

The controller stores the read data into its current write buffer. A drive data function (4 bit code 0110) is sent to the drive after each 16 parcels to acknowledge the received data. When the controller receives a Done pulse from the drive, the write buffer pointer is toggled. The requested data is transferred to the channel loop using Read Buffer Data to Channel function (25x). Upon successful completion, the Drive Status register will contain the value of bus-out that was sent with the read function.

DD-49:

Bus-out data for a read to a DD-49 will contain the following:

Bit 15	Read option bit 3
Bit 14	Read option bit 2
Bit 13	Read option bit 1
Bit 12	Read option bit 0
Bit 11	Zero
Bit 10	Head bit 2
Bit 09	Head bit 1
Bit 08	Head bit 0
Bit 07	Unused
Bit 06	Unused
Bit 05	Sector bit 5
Bit 04	Sector bit 4
Bit 03	Sector bit 3
Bit 02	Sector bit 2
Bit 01	Sector bit 1
Bit 00	Sector bit 0

The bus-out read option bits are defined as follows:

0000	200 - Read Data Sector
0001	201 - Read Sector ID
0010	202 - Read Absolute Data Sector
0011	203 - Read Buffer
0100	204 - Read ECC Block
0101	205 - Read Error Correction Vectors

DS-40:

Bus-out data for a read to a DS-40 will contain the following:

Bit 15	Read option bit 3
Bit 14	Read option bit 2
Bit 13	Read option bit 1
Bit 12	Read option bit 0
Bit 11	Head bit 4
Bit 10	Head bit 3
Bit 09	Head bit 2
Bit 08	Head bit 1
Bit 07	Head bit 0
Bit 06	Unused
Bit 05	Sector bit 5
Bit 04	Sector bit 4
Bit 03	Sector bit 3
Bit 02	Sector bit 2
Bit 01	Sector bit 1
Bit 00	Sector bit 0

The following read options are used on a DS-40 in addition to the above read options for the DD-49:

0110	206 - Read Track header
1000	210 - Read Data Sector, No Continue Read
1010	212 - Read Data Sector from Track Buffer

The full-track buffer in the DS-40 stores a complete track of 48 sectors. When the read-ahead option is enabled during a read operation, the DS-40 is constantly trying to anticipate the data that will be requested next. The most commonly requested sequence is to read from a starting head and sector up to the end of the track, then to increment the head number and read from sector 0 and up in the new track. The DS-40 does this automatically, reading in a full track of data. The read operation stops on the n=1 track buffer holding 48 sectors of data, 12 sectors from each drive in the DS-40 cabinet.

C.11.14 Function 200 - Read Data Sector, Continue Read

This function enables the controller to read ahead a disk sector by queueing a read request and its parameters. If the continue read bit is set, the controller will automatically issue the pending read function and parameters when the present read function is finished. The disk searches for the sector specified on bus-out with the read function. Successful location of the correct sector initiates the data transfer. A complete transfer contains 2048 parcels.

DS-40:

Sector addressing is not consecutive. Addressing is as follows:

Drive	Parameter Sector Value	Absolute Track Buffer Value
A	Sectors 00 - 11	Sectors 00 - 11
B	Sectors 16 - 27	Sectors 12 - 23
C	Sectors 32 - 43	Sectors 24 - 35
D	Sectors 48 - 59	Sectors 36 - 47

C.11.15 Function 201 - Read Sector ID

The specified physical sector ID field is read into the controller buffer. The complete transfer length is 16 parcels.

DD-49:

The ID is encoded in the first 8 parcels of data transferred. If sync cannot be obtained on all four channels to read the ID field, then no data is transferred, an error status is returned, and the Sync Timeout Error flag is set in the drive's General Status register. This occurs when a Read ID command is requested of a sector whose ID was formatted by a Write Defective ID function (302).

Each channel (A1, A2, B1, and B2) has a 32 bit ID field which can be viewed as a vertical string of nibbles n0 through n7. The following table shows the

format of the first 8 transferred parcels. Each nibble of each ID field is represented by its channel number concatenated with the appropriate nibble number. Parcels 8 through 15 will be zero.

Parcel	Bit Positions			
	15-12	11-08	07-04	03-00
0	B2n0	B1n0	A2n0	A1n0
1	B2n1	B1n1	A2n1	A1n1
2	B2n2	B1n2	A2n2	A1n2
3	B2n3	B1n3	A2n3	A1n3
4	B2n4	B1n4	A2n4	A1n4
5	B2n5	B1n5	A2n5	A1n5
6	B2n6	B1n6	A2n6	A1n6
7	B2n7	B1n7	A2n7	A1n7

The format of each nibble in the 32 bit ID field is as follows:

n0	0	h2	h1	h0	Head group number
n1	0	0	c9	c8	Cylinder number bits 9,8
n2	c7	c6	c5	c4	Cylinder number bits 7-4
n3	c3	c2	c1	c0	Cylinder number bits 3-0
n4	0	0	s5	s4	Sector number bits 5,4
n5	s3	s2	s1	s0	Sector number bits 3-0
n6	p7	p6	p5	p4	p7 = -c7
					p6 = -(h2 xor c6)
					p5 = -(h1 xor c5 xor s5)
					p4 = -(h0 xor c4 xor s4)
n7	p3	p2	p1	p0	p3 = -(c3 xor s3)
					p2 = -(c2 xor s2)
					p1 = -(c9 xor c1 xor s1)
					p0 = -(c8 xor c0 xor s0)

DS-40:

The sector ID is encoded in the first 3 parcels of data transferred. The following table shows bit assignments of the 16 parcels received from the drive. Parcels 3 through 15 will be undefined (previous buffer contents).

Parcel	Bit Positions			
	15 14 13 12	11 10 09 08	07 06 05 04	03 02 01 00
0	0 0 0 h4	h3 h2 h1 h0	c11 c10 c9 c8	c7 c6 c5 c4
1	c3 c2 c1 c0	s3 s2 s1 s0	0 0 0 0	0 0 0 0
2	0 0 0 0	d8 d7 d6 d5	d4 d3 d2 d1	d0 p2 p1 p0

where: h4-h0 = Head number
c11-c0 = Cylinder number
s3-s0 = Sector number
d8-d0 = Defect address
p2-p0 = Defect address parity bits

C.11.16 Function 202 - Read Absolute Data Sector

This function is similar to the Read Data Sector function 200. However, this function does not compare ID fields. The sector data is referenced by its absolute position from the index mark. This function is intended as a data recovery technique should an ID field become unreadable. A complete transfer consists of 2048 parcels.

C.11.17 Function 203 - Read Buffer

This function transfers 16 parcels from a DD-49 de-skew buffer, or from the first 16 parcels of a DS-40 track buffer. The contents of bus-out are ignored by the drive. This function is intended as a diagnostic command.

C.11.18 Function 204 - Read ECC Block

This function transfers syndrome data from the drive's syndrome registers. A complete transfer consists of 16 parcels.

DD-49:

The following table lists the encoded syndrome data in the 16 parcels:

Parcel	Contents
1	Channel B2 syndrome bits 47-32

Parcel	Contents
2	Channel B2 syndrome bits 31-16
3	Channel B2 syndrome bits 15-00
5	Channel B1 syndrome bits 47-32
6	Channel B1 syndrome bits 31-16
7	Channel B1 syndrome bits 15-00
9	Channel A2 syndrome bits 47-32
10	Channel A2 syndrome bits 31-16
11	Channel A2 syndrome bits 15-00
13	Channel A1 syndrome bits 47-32
14	Channel A1 syndrome bits 31-16
15	Channel A1 syndrome bits 15-00

Parcels 0,4,8, and 12 will contain zero. This command may be executed by the DD-49 at any time. A syndrome value of zero will be returned for channels that are not flagged in the General Status register as having an ECC error.

DS-40:

Syndrome data is encoded in the first three parcels as follows:

Parcel	Bit Positions			
	15 14 13 12	11 10 09 08	07 06 05 04	03 02 01 00
0	s31 s30 s29 s28	s27 s26 s25 s24	s23 s22 s21 s20	s19 s18 s17 s16
1	s15 s14 s13 s12	s11 s10 s9 s8	s7 s6 s5 s4	s3 s2 s1 s0
2	0 0 0 0	x x x x	x x x x	x x x x

where: s = Syndrome bit
x = Undefined (previous buffer contents)

Parcels 3 through 15 will be undefined (previous buffer contents). The results are only meaningful following a read sector data operation.

C.11.19 Function 205 - Read Error Correction Vectors

This function transfers error correction vectors that are based upon the current contents of the syndrome registers as read by function 204. A complete transfer consists of 16 parcels.

DD-49:

The 16 parcels of error correction data returned are formatted as follows:

Parcel	Bit Positions 15-12	11-08	07-04	03-00	
0	0000	ABCD	EFGH	IJKL	Channel B2 correction mask
1	B2 correction word offset				
2	0000	ABCD	EFGH	IJKL	Channel B1 correction mask
3	B1 correction word offset				
4	0000	ABCD	EFGH	IJKL	Channel A2 correction mask
5	A2 correction word offset				
6	0000	ABCD	EFGH	IJKL	Channel A1 correction mask
7	A1 correction word offset				

Parcels 8 through 15 will be zero. ABCDEFGHIJKL represents a string of 12 bits which contains a 7 bit correction mask. The channel word offset points to the nibble to which the first correction bits ABCD apply. The other correction masks EFGH and IJKL apply to the remaining two nibbles for the associated channel.

The correction bits are logically summed with the appropriate nibble for the channel being corrected. Channel B2 is the most significant nibble (bits 15-12), followed by B1, A2, and A1 in the least significant nibble. An uncorrectable error is indicated by a channel correction word offset value of FFFFh.

This command may be executed by the DD-49 at any time. A syndrome value of zero will be returned for channels that are not flagged in the General Status register as having an ECC error.

DS-40:

The 16 parcels of error correction data returned are formatted as follows:

Parcel	Bit Positions			
	15 14 13 12	11 10 09 08	07 06 05 04	03 02 01 00
0	m7 m6 m5 m4	m3 m2 m1 m0	o15 o14 o13 o12	o11 o10 o9 o8
1	o7 o6 o5 o4	o3 o2 o1 o0	x x x x	x x x x
2	0 0 0 0	x x x x	x x x x	x x x x

where: m = Mask bit
o = Offset bit
x = Undefined (previous buffer contents)

Parcels 3 through 15 will be undefined (previous buffer contents). The results are only meaningful following a read sector data operation.

The offset is counted from the end of the check bits. The most-significant mask bit is applied to the bit at the offset, and the least significant mask bits become the next 7 data bits toward the end of the data field. The bits marked with "x" are the incorrect bits. These bits must be toggled to correct the data field.

The user must determine if (and how much of) the correction mask lies within the data field of the sector that was just read, and apply the correction accordingly. The sector may be uncorrectable (indicated by a correction parcel offset of FFFFh), or the correction may apply (solely or partially) to the syndrome that the correction has already been applied.

C.11.20 Function 206 - Read Track Header, DS-40 Only

This function reads the track-header information that indicates the location of flaws in a track. The function is used in a diagnostic mode. Refer to the "Factory Flaw Track Headers" section of the operator's manual.

C.11.21 Function 210 - Read Data Sector, No Continue Read, DS-40 Only

This function is virtually identical to function 200, except the continue read bit does not apply. A complete transfer contains 2048 parcels.

C.11.22 Function 212 - Read Data Sector from Track Buffer, DS-40 Only

The sector number in function parcel three specifies the logical sector to be read from the track buffer. All 48 sectors in a track are available for reading from the track buffer. The head number in function parcel two is ignored by this function. This function is intended for diagnostic purposes.

C.11.23 Function 25x - Read Buffer Data to Channel

The controller ignores the remaining function parcels and waits to send the function response until the data transfer is complete. Although transfers from the drive will be either 16 or 2048 parcels, channel loop data transfers always consists of eight 256-parcel data blocks. The channel data pulse is blocked during this transfer process.

This controller sends the first channel data pulse to the receiving node followed by a stream of 256 data parcels from the current read buffer. The controller then waits for a channel data pulse from the receiving node to acknowledge the received data block. This process is repeated until all data blocks are sent and the eighth data pulse is received. The controller will then transmit zero response data to the foreground processor to release the channel loop. A call pulse during the data transfer will abort the transfer sequence.

C.11.24 Functions 300-314 - Disk Write Functions

The following write functions send data to the drive:

300	Write Data Sector, Continue Write, 2048 parcels
301	Write Sector ID, 16 parcels
302	Write Defective Sector ID, 16 parcels
303	Write Buffer, 16 parcels
304	Write Data Sector with Zero ECC Block, 2048 parcels
306	Write Track Header, 16 parcels, DS-40 only
310	Write Data Sector, No Continue Write, 2048 parcels, DS-40 only
314	Write Data Sector to Track Buffer, 2048 parcels, DS-40 only

Function parameters were captured when the write function was received. Function parcel two specifies the head group that will be selected after the write operation is complete. Function parcel two has the following format:

Bit 15	Unused
Bit 14	Unused
Bit 13	Unused
Bit 12	Unused
Bit 11	Head bit 4
Bit 10	Head bit 3 / DD-49 Head bit 2
Bit 09	Head bit 2 / DD-49 Head bit 1
Bit 08	Head bit 1 / DD-49 Head bit 0
Bit 07	Head bit 0
Bit 06	Unused
Bit 05	Unused
Bit 04	Unused
Bit 03	Unused
Bit 02	Unused
Bit 01	Unused
Bit 00	Unused

- Bits 11 through 07: These bits specify the logical head group to be selected after this write operation is done.

Function parcel three contains the continue write and last continue write flags (for function 300 only), and the sector number. This parcel has the following bit assignments:

Bit 15	Continue write flag (function 300 only)
Bit 14	Last continue write flag (function 300 only)
Bit 13	Unused
Bit 12	Unused
Bit 11	Unused
Bit 10	Unused
Bit 09	Unused
Bit 08	Unused

Bit 07	Unused
Bit 06	Sector bit 6
Bit 05	Sector bit 5
Bit 04	Sector bit 4
Bit 03	Sector bit 3
Bit 02	Sector bit 2
Bit 01	Sector bit 1
Bit 00	Sector bit 0

- Bit 15: This bit enables the controller to automatically send a write function after the current write operation is complete.
- Bit 14: This bit and bit 15 are set for the write function that is to be the last of a continuous write sequence.
- Bits 06 through 00: The sector number specifies the logical sector to be written. The ID field for this sector should match the current cylinder, current head, and logical sector specified.

The controller receives data from the channel loop before sending the write function to the drive. The controller begins by waiting for a channel data pulse from the transmitting node. Data following the data pulse is written into the current write buffer. A channel data pulse is sent to the transmitting node after 256 parcels have been counted. This process is repeated until eight 256-parcel blocks have been received or a channel call pulse is received. When completed, a write function and its parameters are sent to the drive if it is available. Data to the drive is read from the read buffer pointed to when the write function is sent to the drive.

If the drive is not available after the channel data is received for a continue write, the write function is queued, and sent immediately following completion of the current write operation. Upon successful completion, the Drive Status register will contain the value of bus-out that was sent with the write function.

DD-49:

Bus-out data for a write to a DD-49 will contain the following:

Bit 15	Write option bit 3
Bit 14	Write option bit 2
Bit 13	Write option bit 1
Bit 12	Write option bit 0
Bit 11	Zero

Bit 10	Head bit 2
Bit 09	Head bit 1
Bit 08	Head bit 0
Bit 07	Unused
Bit 06	Unused
Bit 05	Sector bit 5
Bit 04	Sector bit 4
Bit 03	Sector bit 3
Bit 02	Sector bit 2
Bit 01	Sector bit 1
Bit 00	Sector bit 0

The bus-out write option bits are defined as follows:

0000	300 - Write Data Sector
0001	301 - Write Sector ID
0010	302 - Write Defective Sector ID
0011	303 - Write Buffer
0100	304 - Write Data Sector with Zero ECC Block

DS-40:

Bus-out data for a write to a DS-40 will contain the following:

Bit 15	Write option bit 3
Bit 14	Write option bit 2
Bit 13	Write option bit 1
Bit 12	Write option bit 0
Bit 11	Head bit 4
Bit 10	Head bit 3
Bit 09	Head bit 2
Bit 08	Head bit 1
Bit 07	Head bit 0
Bit 06	Unused
Bit 05	Sector bit 5

Bit 04	Sector bit 4
Bit 03	Sector bit 3
Bit 02	Sector bit 2
Bit 01	Sector bit 1
Bit 00	Sector bit 0

The following write options are also used for a DS-40:

0110	306 - Write Track Header
1000	310 - Write Data Sector, No Continue Write
1100	314 - Write Data Sector to Track Buffer

C.11.25 Function 300 - Write Data Sector

This function enables the controller to store write data in the controller while a concurrent write transfer is in progress. The write request and its parameters are queued if the continue write bit is set, and the controller will automatically issue the pending write function when the present write function is done. The disk searches for the sector specified on bus-out with the write function. Successful location of the correct sector initiates the data transfer. A complete transfer contains 2048 parcels.

DS-40:

Sector addressing is not consecutive. See function 200 for DS-40 sector mapping.

C.11.26 Function 301 - Write Sector ID

The first 16 parcels of the controller's current read buffer are written to the specified physical sector ID field.

DD-49:

The ID is encoded in the first 8 parcels of data transferred, and parcels 8 through 15 must be zero. See function 201 for a description of a DD-49 ID field.

DS-40:

The ID is encoded in the first 3 parcels of data transferred, and parcels 3 through 15 must be zero. See function 201 for a description of an ID field.

C.11.27 Function 302 - Write Defective Sector ID

The first 16 parcels of the controller's current read buffer are written to the specified physical sector ID field. These 16 parcels must all be zero. The drive writes these parcels only to the specified sector ID field and sync byte; the sector data field is not modified. A defective ID field is written to avoid accidentally accessing the sector with a read or write request.

C.11.28 Function 303 - Write Buffer

This function transfers 16 parcels to a DD-49 de-skew buffer, or to the first 16 parcels of a DS-40 track buffer. The contents of bus-out are ignored by the drive. The written data can be read by immediately using a Read Buffer function (203). This function is intended as a diagnostic command.

C.11.29 Function 304 - Write Data Sector with Zero ECC Block

This function writes the ECC field following the data field of the specified sector with zeros. The function is part of a diagnostic procedure for exercising the ECC logic. A complete transfer to the drive consists of 16 zero parcels.

C.11.30 Function 306 - Write Track Header, DS-40 Only

Normally not used. Not for diagnostic or software use.

C.11.31 Function 310 - Write Data Sector, No Continue Write, DS-40 Only

This function is virtually identical to function 300, except the continue write flags do not apply. A complete transfer involves 2048 parcels.

C.11.32 Function 314 - Write Data Sector to Track Buffer, DS-40 Only

This function writes a sector of data to the track buffer, but does not transfer the sector to the drive. All 48 sectors in a track are accessible. The head number in function parcel two is ignored by this function. This function is intended as a diagnostics command.

C.11.33 Function 36x - Buffer Echo

The controller ignores function parcels two and three and captures bit 0 from parcel three. Zero response data is immediately sent to the foreground processor. Bit 0 of parcel three selects Buffer A with a zero, and Buffer B with a one value. Both buffer pointers (read and write) are set to the selected buffer.

The controller then waits for a channel data pulse from the transmitting node. Data following the data pulse is written into the selected buffer. A channel data pulse is sent to the transmitting node after 256 parcels have been counted. This process is repeated until eight 256-parcel blocks have been received or a channel call pulse is received. When completed, the contents of the selected echo buffer may be read using the Read Buffer Data to Channel function (25x).

This function is a diagnostic command for checking the controller's buffers. The function sets Buffer Echo mode within the controller that then allows only Read Controller Status (06x), Read Buffer Data to Channel (25x), and Buffer Echo (36x) functions to be accepted. This function does not affect any signals to the drive.

C.11.34 Function 37x - Echo Parameter Word

This function sends an arbitrary value in function parcel three to the drive which is sent back to this controller as well as entered into the drive's echo status register. It is used as a diagnostic function to verify the complete data path to and from the drive. Upon successful completion, the Drive Status register will contain the echoed parcel. The echo parcel can also be read by selecting the drive echo status register using function 04x.

Appendix

D High-Performance Parallel Interface (HIPPI)

D.1 General Description

This stack communicates with a foreground channel loop, a 32-bit High Performance Parallel Interface (HIPPI) channel and a buffer stack. Internal loopback communicates with the corresponding destination stack instead of its external channel. It contains the following registers and counters:

32-bit	General Status register
7-bit	Parity Status register
13-bit	63-entry Packet file
4-bit	End Packet Ranks
16-bit	Call Response register
10-bit	Channel Transfer Length register/counter
8-bit	Channel Block Length counter
12-bit	Channel Buffer Address register
13-bit	Device Buffer Address register
6-bit	Buffer Packet counter
6-bit	Outstanding Readys counter
	16 HIPPI word FIFO buffer

The foreground interface manages data and control with the channel loop and contains the Call Response Register. General status, packet file data, i-field data and buffer read-out data are merged for transmission to the channel loop. Arriving channel data is distributed internally for function parameters and buffer write data.

The buffer stack capacity is 4096 64-bit words, or 32 full-length 32-bit HIPPI bursts. Write data enters the buffer stack from either the channel loop or the FIFO buffer. Buffer read-out data is merged with other data for transmission to the channel loop. The buffer stack may be shared between channel loop and FIFO buffer references.

The FIFO buffer enables both the channel loop and the HIPPI channel to concurrently transfer data. Data streams to a common memory port have an idle window between data blocks, allowing FIFO access to the buffer stack while the recent block is stored in common memory. A minimum capacity of 14 HIPPI words are required in the FIFO to receive two parcels every CLOCK period from the HIPPI channel during a maximum 256-clock-period channel loop data stream out of the buffer stack. The FIFO is emptied into the buffer stack whenever the channel loop is idle and is kept as empty as possible during data transfers.

The controller receives REQUEST, PACKET, BURST, DATA BUS, PARITY BUS, and CLOCK signals from the source, and transmits the CONNECT and READY signals. Departing signals transition on the falling edge of CLOCK, and arriving signals are sampled on the falling edge.

Loopback mode between this controller and the corresponding source controller is enabled if loopback mode from the source controller is set. This stack cannot alter loopback mode. During loopback mode external outputs are deasserted and all external inputs are ignored.

D.1.1 Data Transfer from Source

A typical sequence for a data transfer between a destination and source controller is as follows:

- The destination controller should be cleared and set up for a call response by issuing a Read General Status function. This will cause an interrupt to occur when the source controller asserts the REQUEST signal. The source controller then initiates a connection by asserting REQUEST. The destination responds to REQUEST by sampling the I-Field data that was presented when REQUEST was asserted. If a proper I-Field is received, the destination will issue the answer REQUEST function, which will cause the CONNECT signal to be asserted, thereby establishing a connection between the two controllers. If the destination does not wish to establish a

connection, it may reject the connection by issuing the answer REQUEST function with the reject bit in parcel two set. A rejection will assert CONNECT for only four HIPPI clock periods, which, by convention, signifies a rejection to the source controller.

- Once a connection has been established, the destination controller must issue a function 12 to set the address in the buffer stack that is to begin receiving data. This also provides the call response for a 'packet complete' or 'buffer segment crossed' interrupt.The destination must then pulse the READY signal one or more times to indicate to the source that it is ready to receive data. This is done by issuing a 'Send Ready Pulses' function with the desired number of ready counts. Each ready pulse indicates that the destination is ready to receive 256 HIPPI words. A ready pulse should only be issued when there is enough room in the buffer to receive the full 256 words of a single HIPPI burst. To prevent overflow of the buffer the maximum ready count is 32. The Buffer Packets Counter is incremented for each packet completed and is decremented when the oldest packet entry is read from the Packet File using function 06.
- The source controller will then begin transferring data, holding off additional bursts (256 HIPPI words) until it has received the necessary ready pulses. The destination controller will then receive an interrupt when the source completes its transfer and deasserts the PACKET signal. The destination will also receive an interrupt when the halfway point in the buffer is crossed. This allows for continuous transfer of very long data streams. The controller can be reading data out of the previously filled half of the buffer while the HIPPI channel is transferring data into the other half of the buffer.

The device buffer address, Outstanding Readys Counter, Buffer Packets Counter and Packet File are cleared when CONNECT is asserted. A valid connection is established when CONNECT is asserted for more than 16 CLOCK periods while REQUEST is true.

D.1.2 Transfer Continuation

Once a transfer has been initiated and is under way, it may become necessary to increase the outstanding readys count to keep the transfer going at full speed. The outstanding readys count can be increased by issuing function 13. The difference between READY pulses requested and bursts received is maintained in the Outstanding Readys Counter, to a maximum of 32.

When the Outstanding Readys count is zero and the last HIPPI transfer has completed, function 12 may again set the initial buffer address for arriving HIPPI data.

A connection is terminated when a destination disconnect (function 03) is received. Interrupt response is enabled and CONNECT is automatically deasserted when Packet File Overflow Error, HIPPI Parity Error, LLRC Error, Sequence Error, Long Burst Error, Multiple Short Burst Error, Source Disconnect or FIFO Overflow Error bits become set. Interrupt response is also enabled when REQUEST changes state, the Buffer Segment bit changes, or the Packet File is entered. Buffer Stack Parity Error and Lost Function Error do not enable interrupt response.

D.1.3 Buffer Arbitration

The data buffer is managed such that channel transfers can be going on simultaneously with HIPPI device transfers. This includes the 32-bit read/write functions, as well as channel block transfers to or from the channel loop. The only limitation that must be managed by software is to ensure that the address ranges do not overlap.

D.1.4 64-bit Operation

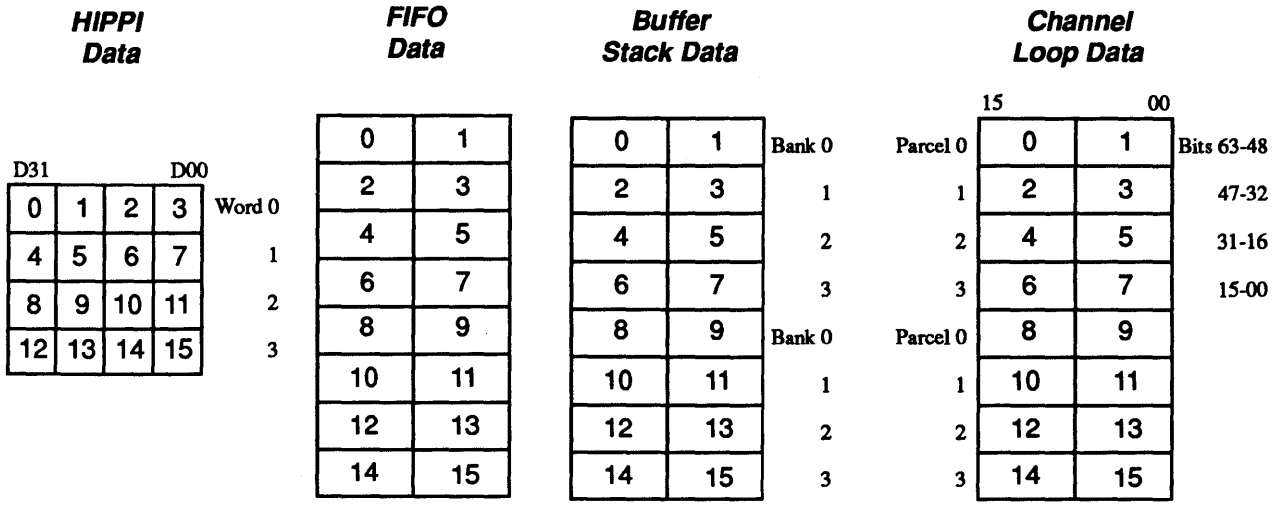
Two 32-bit HIPPI Destination Controllers can be paired together to form a single 64-bit controller. The allowable pairings can be seen on the N Module Board Stack Assignments shown on page four. Both controllers must be separately issued a Clear Controller function with the 64-bit mode bit set. It would be prudent to first clear each controller in 32-bit mode, then to place them each in 64-bit mode with a second clear controller function.

Once both controllers are in 64-bit mode, all functions are issued to the node address of the master controller only. The only exception to this would be when returning the controllers to 32-bit mode. In this case the master should be cleared to 32-bit mode, followed by a clear function to the slave's own node address. A second clear should then be issued again to each controller.

Controller operation in 64-bit mode is otherwise very similar to 32-bit mode. The only difference is that HIPPI words are now 64-bits, and thus the HIPPI device address as referenced by function 12 and the 'end packet buffer address' from function 06 reflect 64-bit word addresses. Note that channel references are for 64-bit words for both 32- and 64-bit operation, and so they operate the same in both modes. Note also that the HIPPI I-Field is defined as 32-bits for both the 32- and 64-bit operation.

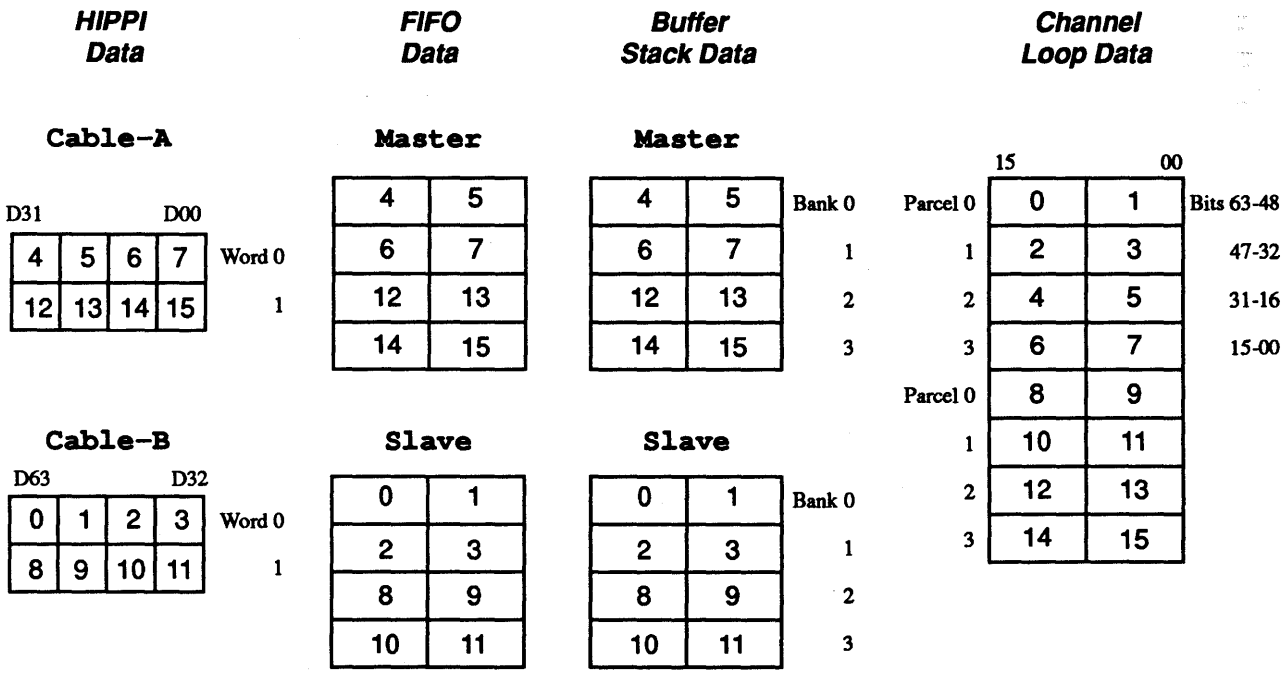
Data byte ordering from the HIPPI channel cable through a single 32-bit controller to the channel loop is illustrated in Figure 6 below.

Figure 6 Data Byte Ordering from HIPPI Channel Cable thru Single 32-bit Controller



Data byte ordering from the HIPPI channel cable through a 64-bit controller to the channel loop is illustrated in Figure 7 below.

Figure 7 Data Byte Ordering from HIPPI Channel Cable thru 64-bit Controller



D.1.5 64-bit Implementation

A 64-bit controller is implemented from two 32-bit controllers by providing both controllers with an identical copy of all HIPPI control signal inputs and

then allowing the controllers to operate in 'lock-step'. Thus, both controllers maintain all the necessary state to sequence themselves properly through the operations rather than have one controller provide the sequencing logic for both. The master controller receives the external 25-MHz HIPPI clock, synchronizes it and passes a synchronized copy to both controllers, timed such that each controller will see the clock edges at exactly the same time. The HIPPI control signals are also received at one point, sampled and fanned out to both controllers with identical timing. This ensures that both controllers see the exact same inputs at the exact same time and thus will operate reliably in lock step.

D.2 General Status Register

This register provides overall status of the controller. Its contents are read to the foreground processor using function 07. The significance of each status bit in the function response data is given below.

Bit 31	Loopback Mode
Bit 30	64-bit Mode
Bit 29	Zero
Bit 28	Zero
Bit 27	Packet File Overflow Error[a]
Bit 26	Sequence Error[a,b]/HIPPI Parity Error[a,c]
Bit 25	LLRC Error[a]
Bit 24	HIPPI Parity Error[b]/Sequence Error[a,c]
Bit 23	Long Burst Error[a]
Bit 22	Multiple Short Burst Error[a]
Bit 21	Source Disconnect[a]
Bit 20	Buffer Stack Parity Error
Bit 19	FIFO Overflow Error[a]
Bit 18	Lost Function Error[a]
Bit 17	State of PACKET
Bit 16	State of CONNECT

Bit 15	State of REQUEST
Bit 14	INTERCONNECT-B in from Cable-B
Bit 13	64-bit Capable
Bit 12	Buffer Segment
Bit 11	Buffer Packet Count bit 5
Bit 10	Buffer Packet Count bit 4
Bit 09	Buffer Packet Count bit 3
Bit 08	Buffer Packet Count bit 2
Bit 07	Buffer Packet Count bit 1
Bit 06	Buffer Packet Count bit 0
Bit 05	Outstanding Readys Count bit 5
Bit 04	Outstanding Readys Count bit 4
Bit 03	Outstanding Readys Count bit 3
Bit 02	Outstanding Readys Count bit 2
Bit 01	Outstanding Readys Count bit 1
Bit 00	Outstanding Readys Count bit 0

a. Reset after the General Status Register is read
b. 64-bit capable controllers only
c. Non 64-bit capable (old) controllers only

D.2.1 Bit 31 - Loopback Mode

Bit 31 indicates that the controller is in internal loopback with the source controller. Clearing the source controller will remove both controllers from loopback mode.

D.2.2 Bit 30 - 64-bit Mode

This bit indicates that the controller is in 64-bit mode and will operate in conjunction with the corresponding master or slave controller to form a complete 64-bit controller. This bit also drives the output signal INTERCONNECT-B on Cable-B. It is set or cleared with the Clear Controller function 00.

D.2.3 Bits 29, 28

Unused (zero)

D.2.4 Bit 27 - Packet File Overflow Error

This bit is set when a packet completes and the Buffer Packet count is 63. Interrupt response is enabled and CONNECT is automatically deasserted when this bit is set. It is reset when the controller is cleared or the General Status register is read.

D.2.5 Bits 26, 24 - Sequence Error/HIPPI Parity Error

The HIPPI Parity error bit is set when any of the four bytes and their corresponding parity bits fail the odd parity test. Interrupt response is enabled and CONNECT is deasserted when this bit is set. It is reset when the controller is cleared or the Parity Status register is read.

This Sequence error bit is set when the signal hierarchy is violated. The cases are: 1) PACKET or BURST true and CONNECT false, 2) PACKET or BURST true and REQUEST false, 3) BURST true and PACKET false, 4) PACKET and BURST become true simultaneously, or 5) PACKET and BURST become false simultaneously and REQUEST is true. Interrupt response is enabled when this bit is set. Cases 2 and 3 will cause CONNECT to be deasserted. The status bit is reset when the controller is cleared or the General Status register is read.

These two bits exchange positions, depending on whether the controller is 64-bit capable or not. This can be determined by bit 13 of the status register.

D.2.6 Bit 25 - LLRC Error

Bit 25 is set when the arriving LLRC at the end of a burst and the accumulated LLRC are not equivalent. Interrupt response is enabled and CONNECT is deasserted when this bit is set. It is reset when the controller is cleared or the General Status register is read.

D.2.7 Bit 24 - Sequence Error/HIPPI Parity Error

See bit 26.

D.2.8 Bit 23 - Long Burst Error

This bit indicates that BURST was true for more than 256 CLOCK periods. Interrupt response is enabled and CONNECT is deasserted when this bit is set. It is reset when the controller is cleared or the General Status register is read.

D.2.9 Bit 22 - Multiple Short Burst Error

This bit is set when a second short burst is completed during a packet. Interrupt response is enabled and CONNECT is deasserted when this bit is set. This bit is reset when the controller is cleared or the General Status register is read.

D.2.10 Bit 21 - Source Disconnect

Bit 21 is set when REQUEST becomes false while CONNECT is asserted. Interrupt response is enabled and CONNECT is deasserted when this bit is set. It is reset when the controller is cleared or the General Status register is read.

D.2.11 Bit 20 - Buffer Stack Parity Error

This bit indicates that a buffer stack read reference experienced a parity error. It is reset when the controller is cleared or the Parity Status register is read. Refer to the function 11 description for controller behavior when a read-out parity error occurs. The state of CONNECT and interrupt response are not affected by this bit.

D.2.12 Bit 19 - FIFO Overflow Error

This bit indicates that the FIFO was holding 31 parcels, the buffer was busy, and another HIPPI word was received. This will occur if a channel data transfer is extended (an internal controller error) beyond the maximum 15.5 words x 40ns/word = 620ns. Interrupt response is enabled and CONNECT is deasserted when this bit is set. It is reset when the controller is cleared or the General Status register is read.

D.2.13 Bit 18 - Lost Function Error

Bit 18 indicates that a function could not be properly executed by the controller. Functions 01 and 02 set this bit if REQUEST is false or CONNECT is asserted. Function 03 is lost if CONNECT is already deasserted. Function 12 will set this bit if the Outstanding Readys count is not zero. Function 13 is lost if CONNECT is not asserted. All functions will be processed during transfer of arriving device data. This bit is reset when the controller is cleared or the

General Status register is read. The state of CONNECT and interrupt response are not affected by this bit.

D.2.14 Bit 17 - State of PACKET

This bit indicates the sampled state of the PACKET signal. PACKET is sampled on the falling edge of CLOCK. Interrupt response is enabled only after the last packet word is in the buffer stack and the Packet File is entered.

D.2.15 Bit 16 - State of CONNECT

This bit indicates the state of the departing CONNECT signal.

D.2.16 Bit 15 - State of REQUEST

Bit 15 indicates the sampled state of the REQUEST signal. REQUEST is sampled on the falling edge of CLOCK. Interrupt response is enabled whenever the REQUEST signal changes state.

D.2.17 Bit 14 - Cable-B Interconnect-in

This bit indicates the sampled state of the HIPPI INTERCONNECT-B signal on cable-B. This status is available only to a master controller.

D.2.18 Bit 13 - 64-bit Capable

This bit indicates that the controller is capable of being paired with another 32-bit controller and operating in 64-bit mode. This also indicates that the controller is a new controller, which has several new features and changes when compared to the old controllers.

D.2.19 Bit 12 - Buffer Segment

Bit 12 is the present state of the device buffer address most-significant bit, indicating which half of the buffer stack will be written with arriving HIPPI data. The controller interrupt is set when this bit changes.

D.2.20 Bits 11 through 06 - Buffer Packet Count

These six bits indicate the number of packets received from the source that have not had their packet status read. This count is incremented after the last packet word is written into the buffer stack. Each completed packet increments

the count by one, to a maximum of 63. It is decremented when an end packet address is read from the Packet File using function 06. The counter is cleared when the controller is cleared or CONNECT becomes asserted.

D.2.21 Bits 05 through 00 - Outstanding Readys Count

These six bits indicate the number of Ready pulses requested, less the number of received bursts. Its value is incremented by the Ready Count when function 13 is received, to a maximum of 32. It is decremented when BURST becomes false. The counter is cleared when the controller is cleared or CONNECT becomes asserted.

D.3 Packet File

This structure is organized as a FIFO, providing a list of buffer addresses that imply the size of each packet received into the destination controller. Each entry gives the address following the last word placed in the buffer for the packet. Thus, by knowing the previous start address the length of the packet can be easily determined.

A new entry is added to the packet file whenever a packet completes. The earliest entry is removed using function 06. A packet file entry can only be read to the foreground processor once, since the read pointer is advanced to the next entry for each function 06. The bit significance of each entry as it is placed in the second parcel of function response data is given below.

Bit 15	Packet File empty
Bit 14	Zero
Bit 13	Zero
Bit 12	End Packet Buffer Address bit 12
Bit 11	End Packet Buffer Address bit 11
Bit 10	End Packet Buffer Address bit 10
Bit 09	End Packet Buffer Address bit 09
Bit 08	End Packet Buffer Address bit 08
Bit 07	End Packet Buffer Address bit 07
Bit 06	End Packet Buffer Address bit 06

Bit 05	End Packet Buffer Address bit 05
Bit 04	End Packet Buffer Address bit 04
Bit 03	End Packet Buffer Address bit 03
Bit 02	End Packet Buffer Address bit 02
Bit 01	End Packet Buffer Address bit 01
Bit 00	End Packet Buffer Address bit 00

D.3.1 Bit 15 - Packet File empty

This bit is set when the Packet File is empty, Buffer Packet Count equal to zero. This value is distinguished from an address entry of zero. The bit will be zero for a valid buffer address value.

D.3.2 Bits 14, 13

Unused (zero)

D.3.3 Bits 12 through 00 - End Packet Buffer Address

These 13 bits contain the device buffer address at the end of a packet. The address points to the HIPPI word following the last word written in the buffer stack. If address bit 00 is found to be set after the end packet address is entered, the device buffer address is advanced one HIPPI word (two parcels) for the beginning of the next packet. The residual HIPPI word at the end of an odd packet will contain arbitrary data.

D.3.4 End Packet Ranks

The End Packet Ranks are comprised of four ranks of registers arranged in a pipeline that contain the data FIFO address at which the last word of a packet was written. This is required to determine at which address in the data buffer the last word of the packet will be written to, and thus which address to save in the packet file described above. Since the packet data is first written to the data FIFO, it is not known which buffer address the data word will be written to. The End Packet Ranks thus provide a flag that identifies the last word of a packet that can be used to know when to capture the buffer address required for the packet file.

The four ranks allow for the unlikely case when four packets are in the data FIFO at one time. In almost all cases only the first rank will ever be used.

D.3.5 Call Response Register

This 16-bit register holds the call response data from parcel one of functions 02, 03, 07 and 13. It is unavailable after the controller is cleared. Its contents will replace the second parcel following a channel call pulse if an interrupt is set in the controller.

D.3.6 Channel Transfer Length Register

This 10-bit register holds the number of 64-bit words to be transferred on the channel loop during functions 10 and 11. Its contents are decremented by 64 when transfer of a 256-parcel data block is complete. A channel loop transfer sequence completes when the value becomes zero. This register is not available to the foreground processor.

D.3.7 Channel Block Length Counter

This eight-bit counter is entered with the number of parcels in a channel loop data block during functions 10 and 11. If the Channel Transfer Length register is greater than or equal to 64 words, this counter is entered with zero (256 parcels). When the Channel Transfer Length register is less than 64 words, the Block Length counter is entered with remaining number of parcels to complete the last data block. A data pulse is expected (function 11) or sent (function 10) when this value decrements to zero.

D.3.8 Channel Buffer Address Register

This 12-bit register captures the buffer word address when a function 10 or 11 is received. The address is sent to the buffer stack when the first data pulse arrives during a function 10.

D.3.9 Device Buffer Address Register

This 13-bit register captures the buffer word address of a function 12. The address is sent to the buffer stack when a new address has been captured or cleared and the first HIPPI word of a packet is written into the buffer stack. The buffer address must reference 32-bit words in 32-bit mode and 64-bit words in 64-bit mode. It is cleared when the controller is cleared or CONNECT becomes asserted.

D.3.10 Buffer Packet Counter

See description in General Status register bits 11 through 06.

D.3.11 Outstanding Readys Counter

See description in General Status register bits 05 through 00.

D.4 Device Signal Descriptions

Refer to the ANSI HIPPI-PH Protocol Specification for complete descriptions and waveform examples of the following signals. The term "rising edge" is used for a logic 0-to-1 transition, and "falling edge" is used for a logic 1-to-0 transition, not necessarily the transitions as seen on an oscilloscope.

The following descriptions apply to both external and internal loopback signals. Loopback output signals to the source controller are always active. During loopback mode external control signals are deasserted and external input signals are ignored. Loopback inputs are ignored during external mode.

D.4.1 INTERCONNECT-A - Input

This signal is received on the translator board only and not made available to the destination controller. The translator board uses the signal to enable all HIPPI control signals to the destination controller. If this signal is inactive, indicating that there is not an available HIPPI source device connected on the other end of the cable, then no activity will be seen by the controller on the HIPPI control signals.

D.4.2 INTERCONNECT-A - Output

This signal forced active on the translator board.

D.4.3 INTERCONNECT-B - Input

This signal indicates that a 64-bit capable source device is connected on the other end of cable-B. It is sampled and made available to the general status register of the master destination controller only.

D.4.4 INTERCONNECT-B - Output

This signal indicates that the destination controller is set up in 64-bit mode. It is set or cleared by a Clear Controller function 00.

D.4.5 REQUEST - Input

This signal becomes true when the source initiates a connection. The arriving signal is sampled on the falling edge of CLOCK. A connection is established if CONNECT is asserted for more than 16 CLOCK periods and REQUEST is true. Interrupt response is enabled when REQUEST changes state.

D.4.6 PACKET - Input

This signal delimits a group of data bursts as a packet. It is sampled on the falling edge of CLOCK. When PACKET becomes false, an end packet mark is queued in the end packet ranks. The interrupt response is enabled only after the last word of the packet is written to the buffer stack and the buffer stack device address is entered into the Packet File.

D.4.7 BURST - Input

This signal delimits a group of words on the DATA BUS as a burst. A burst may contain one to 256 words followed by an LLRC word, 80ns to 10.28 s. BURST is sampled on the falling edge of CLOCK if PACKET is true. The LLRC data word is sampled in the CLOCK period following BURST becoming false, and the Outstanding Readys Counter is decremented.

D.4.8 DATA BUS - Inputs, 32 bits

These bits carry a 32-bit i-field, data word or LLRC from the source device. The i-field data is captured for response data when function 01 is received. Data words are sampled on the falling edge of CLOCK when BURST is true. The LLRC word is sampled on the falling edge of CLOCK in the clock period immediately after BURST becomes false.

D.4.9 PARITY BUS - Inputs, 4 bits

Four parity bits are received from the source with each 32-bit word. Each parity bit is associated with a byte of input data as follows:

Parity bit P3, byte 0	Data bits D31 through D24
P2, byte 1	Data bits D23 through D16
P1, byte 2	Data bits D15 through D08
P0, byte 3	Data bits D07 through D00

Each parity bit and its associated byte must have odd parity to avoid a HIPPI parity error. Timing for these signals is the same as the data signals.

D.4.10 CLOCK - Input

This signal is used for synchronization of control and data signals from the source device. Transitions are detected after resynchronization to the Cray-3 system clock. The REQUEST, PACKET, BURST, DATA BUS and PARITY BUS signals will be sampled on the detected falling edge of CLOCK.

D.4.11 CONNECT - Output

This signal is used to maintain or reject a connection with the source. The signal transitions on the falling edge of CLOCK. Function 01 is used to capture the i-field data before CONNECT is asserted. If function 02 is received with the reject bit set, CONNECT will be asserted for only four CLOCK periods. CONNECT is deasserted when REQUEST becomes false, function 03 is received, or a burst error occurs. Asserting CONNECT will clear the buffer device address, Outstanding Readys Counter, pending READY count, Buffer Packet Counter and the Packet File.

D.4.12 READY - Output

This signal indicates that the controller is prepared to accept data, one burst for each Ready pulse sent. Each READY pulse is asserted for only four CLOCK periods on the falling edge of CLOCK. Pulses will not be sent unless CONNECT has been asserted for at least four CLOCK periods.

D.5 Controller Functions

The HIPPI input controller recognizes a unique identification code in the first data parcel following a channel function pulse. When this identification code matches the node's internal value, the function parcels may be captured by the controller. The first parcel contains the following fields which are examined by the controller:

Bits 15-13	Three bits, 101 for HIPPI destination nodes
Bits 12-09	Four bits, node address
Bits 08-04	Five bits, ignored
Bits 03-00	Four bits, function code

A function is identified as directed to this node when bits 15 through 09 match the node values. The function field is then translated for the specific action requested by the foreground processor. These functions are listed below.

D.6 HIPPI Destination Function Summary

Control Functions:

00 - Clear Controller

01 - Sample I-field

02 - Answer REQUEST

03 - Deassert CONNECT

04 - Read Parity Status

05 - Unused

06 - Read Packet File

07 - Read General Status

Data Functions:

10 - Transfer Data from Channel Loop to Buffer Stack

11 - Transfer Data to Channel Loop from Buffer Stack

12 - Set Device Buffer Address

13 - Send Ready Pulses

14 - Read Lower 32 Bits from Buffer Stack to Foreground

15 - Read Upper 32 Bits from Buffer Stack to Foreground

16 - Write Lower 32 Bits from Foreground to Buffer Stack

17 - Write Upper 32 Bits from Foreground to Buffer Stack

Unused functions will not respond with a function response. This will leave the channel loop busy whereby the foreground processor can detect a channel busy time-out. The following table shows the HIPPI destination controller function parameters and responses, along with the status register bit assignments.

D.6.1 Function Table

D.6.2 Function Descriptions

D.6.2.1 Function 00 - Clear Controller, Set Mode

The controller responds immediately with zero response data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - ignored
- Parcel 2 - mode request
- Parcel 3 - ignored

The controller is cleared as if dead start was received. This involves deasserting the CONNECT and READY signals, resetting mode flags and clearing all counters, the General Status register and the Packet File. The controller will not respond to a channel call until a function 02, 03, 07 or 13 is received. However, interrupt conditions received before interrupts are enabled will set the interrupt response flag.

The mode request parcel is used to set or clear 64-bit mode by setting or clearing bit 14.

If this stack is in loopback mode with the source stack, it will remain in loopback mode after this function, since the source controller controls loopback mode. All external signals to the destination are deasserted, and all external signals are ignored during loopback.

D.6.2.2 Function 01 - Sample I-field

The controller responds immediately with conditional response data to the foreground processor. The last three function parcels are ignored:

- Parcel 1 - ignored
- Parcel 2 - ignored
- Parcel 3 - ignored

This function sets Lost Function Error if REQUEST is false or CONNECT is asserted. The controller returns the sampled i-field data only if REQUEST is true and CONNECT is deasserted. If the function is 'lost', the controller responds immediately with zero response data.

D.6.2.3 Function 02 - Answer REQUEST

The controller responds immediately with zero data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - call response data
- Parcel 2 - reject connection in bit 15
- Parcel 3 - ignored

If the reject bit is reset, CONNECT remains asserted for a connection. If the Reject bit is set, CONNECT is asserted for only four CLOCK periods and then deasserted. This function sets Lost Function Error if REQUEST is false or CONNECT is asserted.

D.6.2.4 Function 03 - Deassert CONNECT

The controller responds immediately with zero data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - call response data
- Parcel 2 - ignored
- Parcel 3 - ignored

The CONNECT signal is deasserted in the next CLOCK period to disconnect the HIPPI channel. This function is always accepted while CONNECT is asserted to allow an unconditional disconnect. Lost Function Error is set if CONNECT is already deasserted. The CONNECT signal is automatically deasserted if REQUEST becomes false or certain error conditions are detected. The controller interrupt is enabled when REQUEST becomes false, indicating that the source has also disconnected.

D.6.2.5 Function 04 - Read Parity Status

The controller responds immediately with the following data:

Bits	Description
31-27	unused - zero
26	HIPPI parity error occurred
25-24	HIPPI byte with parity error 00 - bits 07-00 01 - bits 15-08 10 - bits 23-16 11 - bits 31-24
23	Buffer parity error occurred

Bits	Description
22-20	Buffer byte with parity error 000 - bits 63-56 001 - bits 55-48 010 - bits 47-40 011 - bits 39-32 100 - bits 31-24 101 - bits 23-16 110 - bits 15-08 111 - bits 07-00
19-00	Unused - zero

The parity status register above is cleared when read. The HIPPI parity error and buffer parity error bits in the status register are also cleared when the parity status register is read.

D.6.2.6 Function 05

Ignored (no response)

D.6.2.7 Function 06 - Read Packet File

The controller responds with the earliest Packet File entry or Packet File empty value to the foreground processor. The last three function parcels are ignored:

- Parcel 1 - ignored
- Parcel 2 - ignored
- Parcel 3 - ignored

The oldest entry is returned in the second parcel of the response data. The first response parcel will contain zero. The Packet File read pointer is then advanced to the next entry. The Packet File empty value is returned if the Buffer Packet count is zero (bit 15 set).

D.6.2.8 Function 07 - Read Status

The controller responds with the General Status register to the foreground processor. The last three function parcels will contain the following information:

- Parcel 1 - call response data
- Parcel 2 - ignored
- Parcel 3 - ignored

General Status bits 31-16 are placed in the first response parcel, and bits 15-00 are placed in the second parcel. Reading the General Status register will reset Source Disconnect and all error flags after the register is read.

D.6.2.9 Function 10 - Transfer Data from Channel Loop to Buffer Stack

The controller responds immediately with zero data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - 12-bit buffer word write address
- Parcel 2 - ignored
- Parcel 3 - 10-bit channel transfer length

The write address is held until the first data block arrives. If the transfer length is greater than 64 words, the transfer is divided into 256-parcel segments. The arriving channel data pulse is blocked during the transfer process. This function can be processed during a device data transfer since buffer stack data from the source is buffered in a FIFO buffer.

The controller then waits for a channel data pulse from the transmitting node. The write address is sent to the buffer stack when the first data pulse arrives. Data following the data pulse is written into the buffer up to 256 parcels. A channel data pulse is sent to the transmitting node when either 256 parcels have been received or the Channel Transfer Length register has decremented to zero. This process is repeated until the Channel Transfer Length becomes zero or a channel call pulse is received. If a zero transfer length is received with the function, no data is expected and the controller completes the function immediately.

D.6.2.10 Function 11 - Transfer Data to Channel Loop from Buffer Stack

The controller captures the following parcels, but a function response is not sent immediately. The last three function parcels will contain the following:

- Parcel 1 - 12-bit buffer word read address
- Parcel 2 - ignored
- Parcel 3 - 10-bit channel transfer length

The read address is sent directly to the buffer stack to begin the first read-out block. If the transfer length is greater than 64 words, the transfer is divided into 256-parcel segments. The arriving channel data pulse is blocked during the transfer process. If a zero transfer length is received with the function, no data is transmitted and the controller completes the function immediately. This function can be processed during a device data transfer since departing data is buffered through a FIFO.

A channel data pulse is sent to the receiving node, followed by up to 256 parcels from the buffer. The controller then waits for a channel data pulse from the receiving node to acknowledge the data block. This process is repeated until the transfer length is decremented to zero and the final expected data pulse is received or a call pulse is received. If a short data block (less than 64 words) is necessary, it is sent in the final block. The controller transmits zero response data to the foreground processor after the entire sequence is complete.

If a parity error occurs during the buffer read-out data stream to the channel loop, a Buffer Byte Parity Error flag will be set in the General Status register and the transfer sequence will continue.

D.6.2.11 Function 12 - Set Device Buffer Address

The controller responds immediately with zero data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - 13-bit HIPPI word buffer write address
- Parcel 2 - ignored
- Parcel 3 - ignored

This function allows the buffer location of the initial packet word to be set. The Device Buffer Address register is cleared when the controller is cleared or CONNECT becomes asserted. This function is not accepted (sets Lost Function Error) if the Outstanding Readys count is not zero.

D.6.2.12 Function 13 - Send Ready Pulses to Source

The controller responds immediately with zero data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - call response data
- Parcel 2 - ignored
- Parcel 3 - six-bit additional ready count

The ready count, lowest six bits in function parcel 3, is added to an internal Pending Ready counter and the Outstanding Readys counter, up to a maximum of 32. READY pulses are sent to the source until the Pending Ready counter is decremented to zero. This function is not accepted (sets Lost Function Error) if CONNECT is not asserted.

D.6.2.13 Function 14 - Read Lower 32 Bits to Foreground

The controller responds with buffer read-out data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - 12-bit buffer word read address
- Parcel 2 - ignored
- Parcel 3 - ignored

Response data contains read-out bits 31-16 in the first parcel and bits 15-00 of the addressed word in the second parcel. This function can be accepted during an active device transfer.

D.6.2.14 Function 15 - Read Upper 32 Bits to Foreground

The controller responds with buffer read-out data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - 12-bit buffer word read address
- Parcel 2 - ignored
- Parcel 3 - ignored

Response data contains read-out bits 63-48 in the first parcel and bits 47-32 of the addressed word in the second parcel. This function can be accepted during an active device transfer.

D.6.2.15 Function 16 - Write Lower 32 Bits to Buffer Stack

The controller responds immediately with zero data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - 12-bit buffer word write address
- Parcel 2 - Write data bits 31-16
- Parcel 3 - Write data bits 15-00

D.6.2.16 Function 17 - Write Upper 32 Bits to Buffer Stack

The controller responds immediately with zero data to the foreground processor. The last three function parcels will contain the following:

- Parcel 1 - 12-bit buffer word write address
- Parcel 2 - Write data bits 63-48
- Parcel 3 - Write data bits 47-32

Index

A

B

C

N

P

R

S

T

V

W