CRAY

RESEARCH, INC.

CRAY-1®

COMPUTER SYSTEMS

HARDWARE REFERENCE MANUAL

HR-0004

Revision	Description
	January 1976 - Original printing
A	May 1976 - Reprint with revision
A-01	September 1976 - Corrections to pages 3-20, 3-27, 4-9, 4-10, 4-28, 4-36, 4-43, 4-55, and 4-57.
B	October 1976 - Reprint with revision. Addition of floating point range error detection, vector floating point error, and error correction.
B-01	February 1977 - Changes to exchange pckage, additions to instructions 152 and 153, corrections to syndrom bit description, corrections to instruction summary, appendix D.
B-02	July 1977 - Corrections and changes to pages xi, 2-3, 3-19 through 3-28.1, 3-31, 3-34, 3-36, 3-38, 4-14 through 4-17, 4-54, 4-68, 5-1, 5-3, 5-4, 5-6, 6-2, A-4, D-1 through D-4.
C	November 1977 - This printing obsoletes revision B. Features added include 8-bank phasing and I/O master clear procedure. Chart tape reflects only changes introduced with this revision.
C-01	April 1978 - This change packet changes the nomenclature for two flags in the exchange package (page 3-37) and corrects technical errors on pages 2-11, 4-71, 5-6, 6-5, and A-3.
C-02	July 1978 - This change packet documents changes to the physical description of the CRAY-1 Computer System. Changes are all in section 2.
D	August 1978 - This printing is exactly the same as revision C with change packets C-01 and C-02 incorporated.

Revision	Description

E   May 15, 1979 - Reprint with revision. This printing corrects
the description of the multiply algorithm and adds descriptions
of various standard options (i.e., vector population
instructions, programmable clock interrupt, and monitor mode
interrupt). In addition, sections 5 and 6 have been rewritten.
Revision E obsoletes versions C and D of this publication.

E-01 May, 1980 - This change packet documents changes to the multiply
functional unit that supports symmetrical multiply, documents
CAL instruction changes, and corrects miscellaneous technical
errors. Changes are noted by change bars.

F   May, 1982 - This reprint with revision incorporates revision E
with E-01. With this printing, the publication number has been
changed to HR-0004. No other changes have been made.

CONTENTS

APPENDIXES

FIGURES

TABLES

SECTION 1

INTRODUCTION

The CRAY-1 Computer System is a powerful general-purpose computer capable of extremely high processing rates. These rates are achieved by combining scalar and vector capabilities into a single central processor which is joined to a large, fast, bi-polar memory. Vector processing by performing iterative operations on sets of ordered data provide results at rates greatly exceeding result rates of conventional scalar processing. Scalar operations complement the vector capability by providing solutions to problems not readily adapted to vector techniques.

Figure 1-1 represents the basic organization of a CRAY-1 system. The central processor unit (CPU) is a single integrated processing unit consisting of a computation section, a memory section, and an input/ output section. The memory is expandable from 0.25 million 64-bit words to a maximum of 1.0 million words. The 12 input channels and 12 output channels in the input/output section connect to a maintenance control unit (MCU), a mass storage subsystem, and a variety of front-end systems or peripheral equipment. The MCU provides for system initializa- tion and for monitoring system performance. The mass storage subsystem provides secondary storage and consists of one to eleven Cray Research DCU-2 Disk Controllers, each with one to four DD-19 Disk Storage Units. Each DD-19 has a capacity of 2.424×10^9 bits.

I/O channels can be connected to independent processors referred to as front-end computers or I/O stations or can be connected to peripheral equipment according to the requirements of the individual installation. At least one front-end system is considered standard to collect data and present it to the CRAY-1 for processing and to receive output from the CRAY-1 for distribution to slower devices.

Table 1-1 summarizes the characteristics of the system. The following paragraphs provide an additional introduction to the three sections of the CPU; later sections of this manual describe the features in detail.
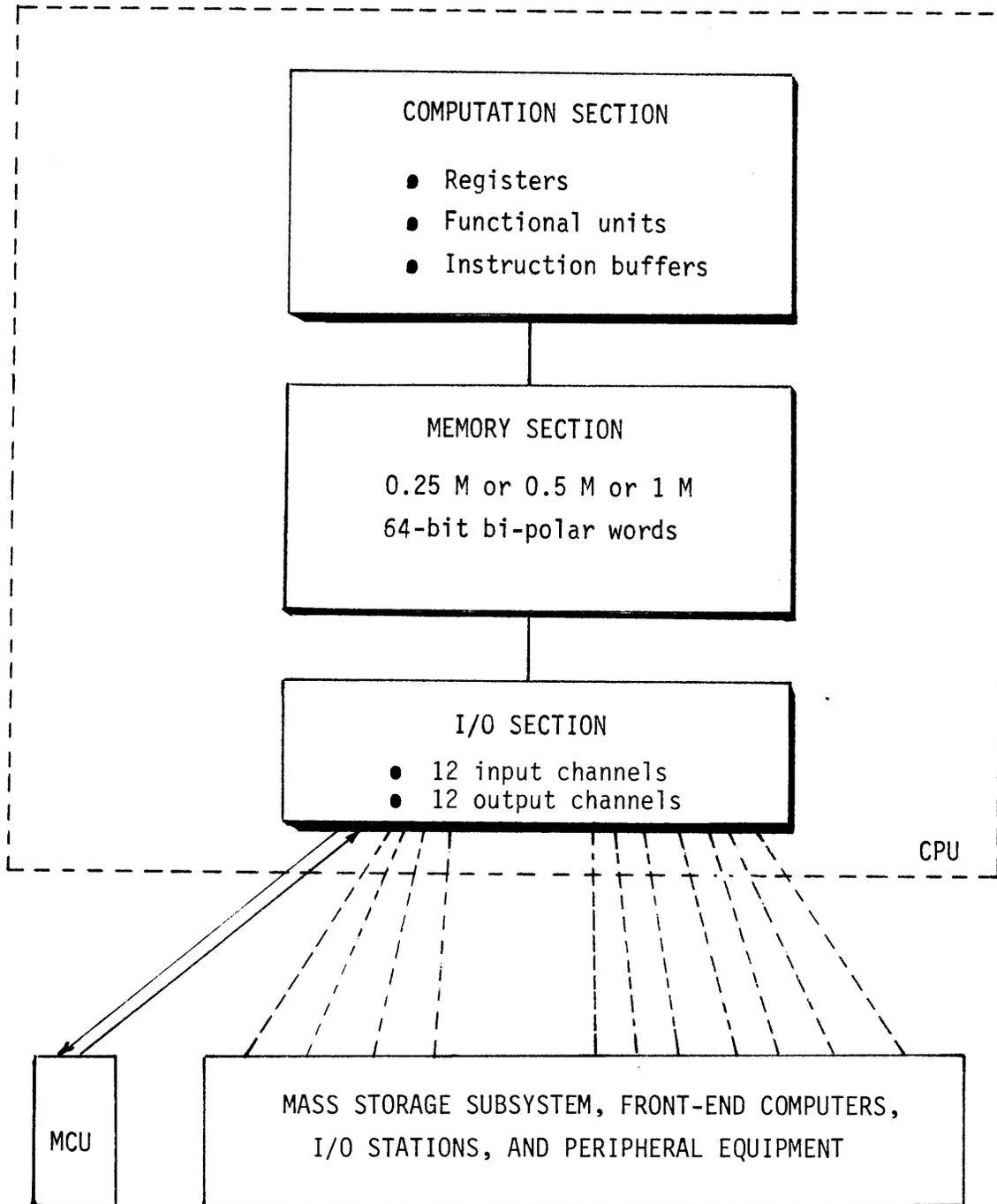
Figure 1-1. Basic computer system

Table 1-1. Characteristics of the CRAY-1 Computer System

COMPUTATION SECTION

- 64-bit word
- 12.5 nanosecond clock period
- 2's complement arithmetic
- Scalar and vector processing modes
- Twelve fully segmented functional units
- Eight 24-bit address (A) registers
- Sixty-four 24-bit intermediate address (B) registers
- Eight 64-bit scalar (S) registers
- Sixty-four 64-bit intermediate scalar (T) registers
- Eight 64-element vector (V) registers, 64-bits per element
- Four instruction buffers of 64 16-bit parcels each
- Integer and floating point arithmetic
- 128 instruction codes

MEMORY SECTION

- Up to 1,048,576 words of bi-polar memory
  (64 data bits and eight error correction bits)
- Eight or sixteen banks
- Four-clock-period bank cycle time
- One word per clock period transfer rate to B, T, and V registers
- One word per two clock periods transfer rate to A and S registers
- Four words per clock period transfer rate to instruction buffers
- Single error correction - double error detection (SECDED)

INPUT/OUTPUT SECTION

- Twelve input channels and twelve output channels
- Channel groups contain either six input or six output channels
- Channel groups served equally by memory (scanned every four clock periods)
- Channel priority resolved within channel groups
- Sixteen data bits, three control bits per channel, four parity bits, and an external master clear
- Lost data detection

COMPUTATION SECTION

The computation section contains instruction buffers, registers and
functional units which operate together to execute a program of
instructions stored in memory.

Arithmetic operations are either integer or floating point. Integer
arithmetic is performed in two's complement mode. Floating point
quantities have signed-magnitude representation.

The CRAY-1 executes 128 operation codes as either 16-bit (one parcel) or
32-bit (two-parcel) instructions. Operation codes provide for both
scalar and vector processing.

Floating point instructions provide for addition, subtraction, multi-
plication, and reciprocal approximation. The reciprocal approximation
instruction allows for the computation of a floating divide operation
using a multiple instruction sequence.

Integer or fixed point operations are provided as follows: integer
addition, integer subtraction, and integer multiplication. An integer
multiply operation produces a 24-bit result; additions and subtractions
produce either 24-bit or 64-bit results. No integer divide instruction
is provided and the operation is accomplished through a software
algorithm using floating point hardware.

The instruction set includes Boolean operations for OR, AND, and exclusive
OR and for a mask-controlled merge operation. Shift operations allow the
manipulation of either 64-bit or 128-bit operands to produce 64-bit
results. With the exception of 24-bit integer arithmetic, all operations
are implemented in vector as well as scalar instructions. The integer
product is a scalar instruction designed for index calculation. Full
indexing capability allows the programmer to index throughout memory in
either scalar or vector modes. The index may be positive or negative in
either mode. This allows matrix operations in vector mode to be performed
on rows or the diagonal as well as conventional column-oriented operations.

Each functional unit implements an algorithm or a portion of the instruction
set. Units are independent and are fully segmented. This means that a new
set of operands for unrelated computation may enter a functional unit each
clock period.

MEMORY SECTION

The memory for the CRAY-1 normally consists of 16 banks† of bi-polar
LSI memory. Three memory size options are available: 262,144 words,
524,288 words, or 1,048,576 words. Each word is 72 bits long and consists
of 64 data bits and 8 check bits. The banks are independent of each other.

Sequentially addressed words reside in sequential banks. The memory cycle
time is four clock periods (50 nsec). The access time, that is, the time
required to fetch an operand from memory to a scalar register is 11 clock
periods (137.5 nsec).

The maximum transfer rate for B, T, and V registers is one word per
clock period. For A and S registers, it is one word per two clock
periods. Transfers of instructions to the instruction buffers occur
at a rate of 16 parcels (four words) per clock period.

Thus, the high speed of memory supports the requirements of scientific
applications while its low cycle time is well suited to random access
applications. The phased memory banks allow high communication rates
through the I/O section and provide low read/store times for vector
registers.

INPUT/OUTPUT SECTION

Input and output communication with the CRAY-1 is over 12 full duplex
16-bit channels. Associated with each channel are control lines that
indicate the presence of data on the channel (ready), data received
(resume), or transfer complete (disconnect).

The channels are divided into four channel groups. A channel group
consists of either six input paths or six output paths. The four
channel groups are scanned sequentially for I/O requests at a rate of
one channel group per clock period. The channel group will be reinterrogated
four clock periods later whether any I/O request is pending in the channel
or not. If more than one channel of the channel group is active, the
requests are resolved on a priority basis. The request from the lowest
numbered channel is serviced first.

† See 8-Bank Phasing Option, section 5.

VECTOR PROCESSING

All operands processed by the CRAY-1 are held in registers prior to their
being processed by the functional units and are received by registers
after processing. In general, the sequence of operations is to load one
or more vector registers from memory and pass them to functional units.
Results from this operation are received by another vector register and
may be processed additionally in another operation or returned to memory
if the results are to be retained.

The contents of a V register are transferred to or from memory by
specifying a first word address in memory, an increment for the memory
address, and a length. The transfer proceeds beginning with the first
element of the V register and incrementing by one in the V register at
a rate of up to one word per clock period depending on memory conflicts.

A result may be received by a V register and re-entered as an operand to
another vector computation in the same clock period. This mechanism
allows for "chaining" two or more vector operations together. Chain
operation allows the CRAY-1 to produce more than one result per clock
period. Chain operation is detected automatically by the CRAY-1 and
is not explicitly specified by the programmer, although the programmer
may reorder certain code segments in order to enable chain operation.

There may be a conflict between scalar and vector operations only for the
floating point operations and storage access. With the exception of these
operations, the functional units are always available for scalar operations.
A vector operation will occupy the selected functional unit until the
vector has been processed.

Parallel vector operations may be processed in two ways:
   1. Using different functional units and all different V registers.
   2. Chain mode, using the result stream from one vector register
       simultaneously as the operand to another operation using a
       different functional unit.

Parallel operations on vectors allow the generation of two or more results
per clock period. Most vector operations use two vector registers as

operands or one scalar and one vector register as operands. Exceptions are vector shifts, vector reciprocal, and the load or store instructions.

Since many vectors exceed 64 elements, a long vector is processed as one or more 64-element segments and a possible remainder of less than 64 elements. Generally, it is convenient to compute the remainder and process this short segment before processing the remaining number of 64-element segments; however, a programmer may choose to construct the vector loop code in any of a number of ways. The processing of long vectors in FORTRAN is handled by the compiler and is transparent to the programmer.

SECTION 2

PHYSICAL ORGANIZATION
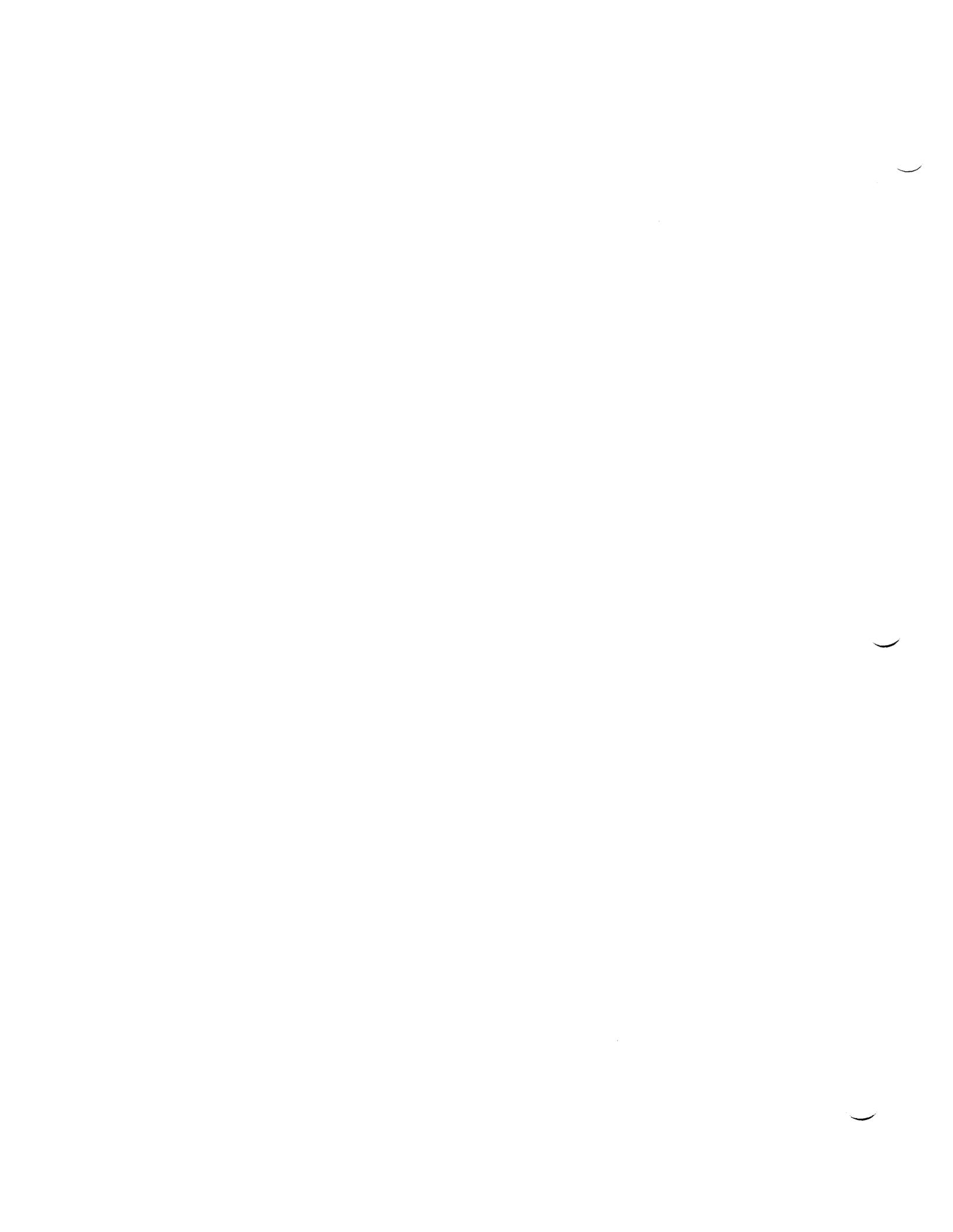
INTRODUCTION

The CRAY-1 computer system consists of the following:
- The CPU mainframe
- A power cabinet
- A condensing unit
- Two motor generators and control cabinets
- A maintenance control unit (MCU)
- One or more disk systems, and
- Optional interfaces to one or more front-end computer systems.

MAINFRAME

The CRAY-1 mainframe, figure 2-1, is composed of 24 logic chassis. The
chassis are arranged two per column in a 270° arc which is about five feet
in diameter. The twelve columns are about 6 1/2 ft tall. At the base of
the columns, 1 1/2 ft high and extending outward about 2 1/2 ft, are
cabinets for power supplies and cooling distribution systems.

Viewing the cabinet from the top, the chassis of the upper circle are labeled
A through L proceeding in a counter-clockwise direction from the opening.
The chassis of the lower circle are labeled M through X. The assignment
of modules to chassis is illustrated in figure 2-2.

MODULES

The CRAY-1 computer system uses only one basic module construction through-
out the entire machine. The module consists of two 6 x 8 inch printed
circuit boards mounted on opposite sides of a heavy copper heat transfer
plate. Each printed circuit board has capacity for a maximum of 144
integrated circuit (IC) packages and approximately 300 resistor packages.
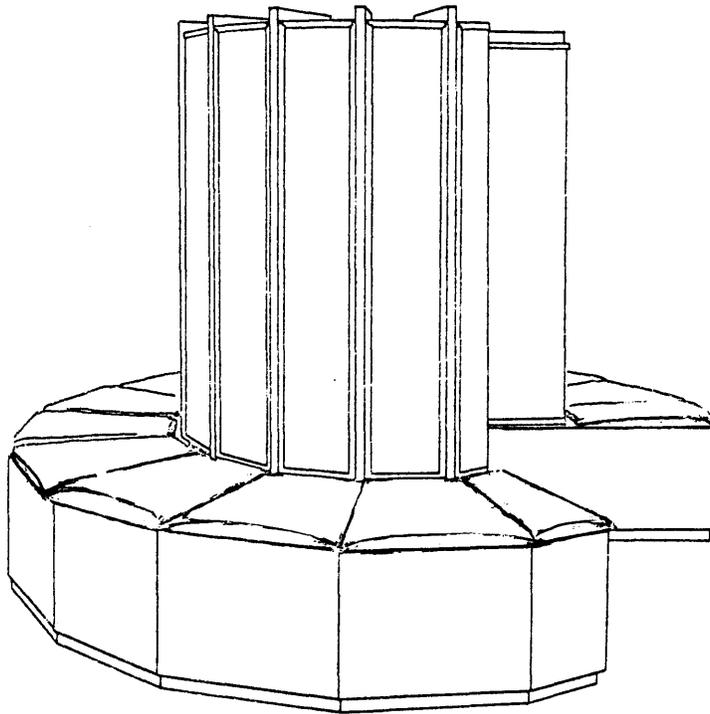
- Dimensions

    Base - approximately 9 ft diameter by 1 1/2 ft high

    Columns - approximately 5 ft diameter by 6 1/2 ft high including
        height of base

- 24 chassis arranged two per column in 12 columns
- Approximately 1700 modules (16 banks); approx. 115 standard module types
- Each module contains up to 288 IC packages per module
- Power consumption approximately 118 kw input for maximum memory size
- Refrigerant-22 cooled with refrigerant/water heat exchange
- Three memory options
- Weight 10,500 lbs (maximum memory size)
- Three basic chip types

    5/4 NAND gates

    Memory chips

    Register chips

Figure 2-1. Physical organization of mainframe
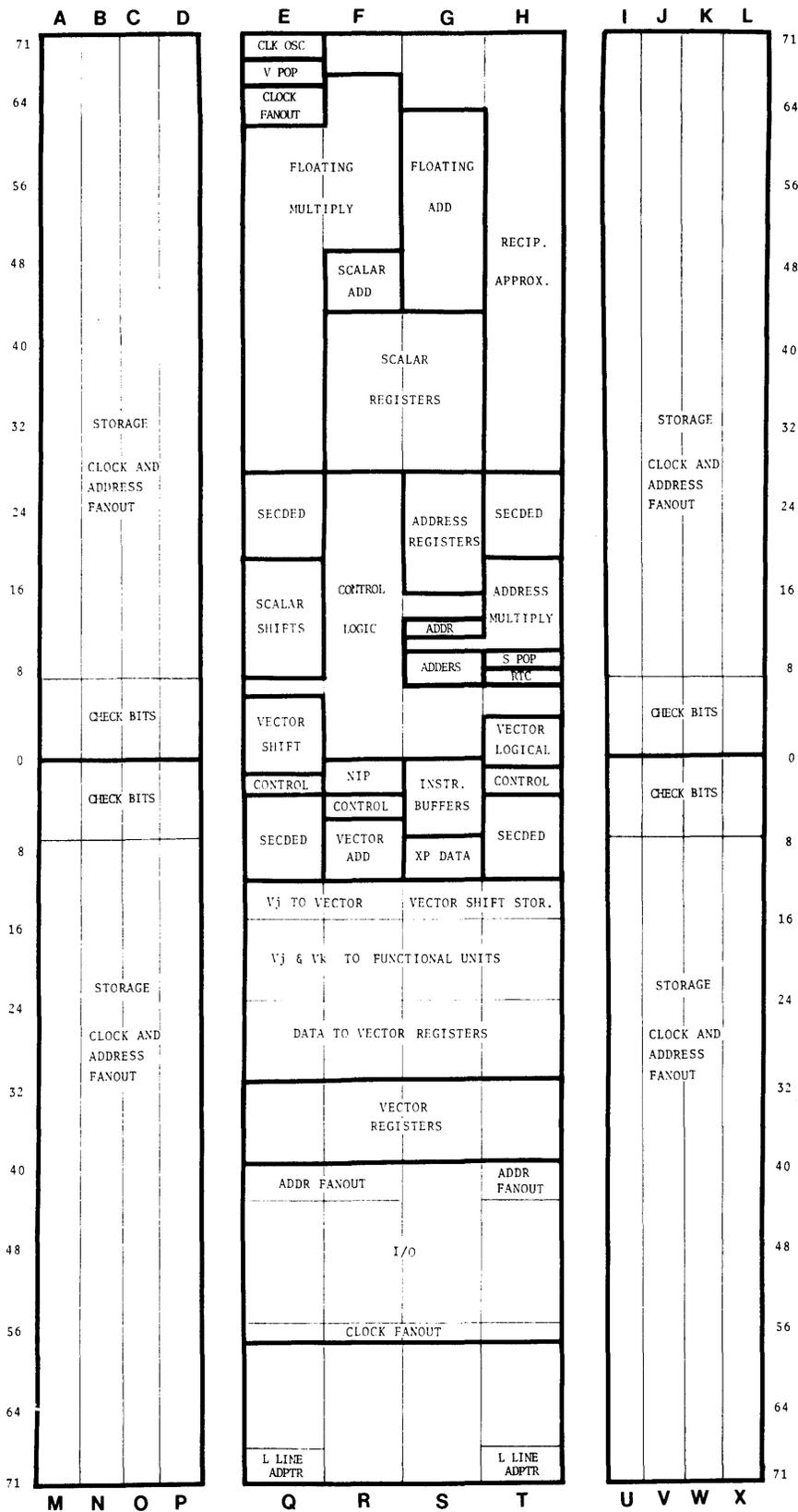
Figure 2-2. General chassis layout

There are 1662 modules in a CRAY-1 with a standard 16-bank[†] memory. Modules are arranged 72 per chassis as illustrated in figure 2-2. There are over 115 module types. Usage varies from 1 to over 700 modules per type. Module type and usage is summarized in Appendix B. Each module type is identified by two letters. The first indicates the module series (A, D, F, G, H, J, M, R, S, T, V, X, and Z). The second letter identifies types of modules within a series.

The computation and I/O modules are on the eight chassis forming the center four columns. Each of the eight chassis on either side of the four center columns contains one of the 16 memory banks.

Modules are cooled by transferring heat via the heat transfer plate to cooling bars which in turn transfer the heat to a refrigerant-22. Power dissipation depends on module density. The average module dissipation by usage is approximately 50 watts.

Two supply voltages are used for each module: -5.2 volts for IC power; -2.0 volts for line termination.

Each module has 96 pin pairs available for interconnecting to other modules. All interconnections are via twisted pair wire. The average utilization of pins is approximately 60 percent.

Each module has 144 available test points that can be used for trouble shooting. Test points are driven by circuits that do not drive other loads.

CLOCK
All timing within the mainframe cabinet is controlled by a single phase synchronous clock network. This clock has a period of 12.5 nsec. The lines that carry the clock signal from the central clock source to the individual modules of the CPU are all made of uniform length so that the leading edge of a clock signal arrives at all parts of the CPU cabinet at the same time. A three nanosecond pulse (figure 2-3) is formed on each module.
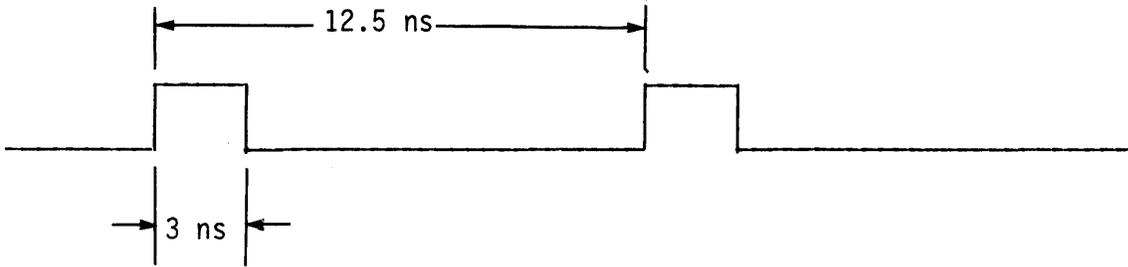
Figure 2-3. Clock pulse waveform

References to clock periods in this manual are often given in the form
CPn where n indicates the number of the clock period during which an
event occurs. Clock periods are numbered beginning with CP0. Thus, the
third clock period would be referred to as CP2.

POWER SUPPLIES

Thirty-six power supplies are used for the CRAY-1 computer system. There
are twenty -5.2 volt supplies and sixteen -2.0 volt supplies. The supplies
are divided into twelve groups of three. Each group supplies one column.

The power supply design assumes a constant load. The power supplies do not
have internal regulation but depend on the motor-generator to isolate and
regulate incoming power. The power supplies use a twelve-phase transformer,
silicon diodes, balancing coil, and a filter choke to supply low ripple
DC voltages. The entire supply is mounted on a refrigerant-22 cooled heat
sink. Power is distributed via bus bars to the load.

PRIMARY POWER SYSTEM

The primary power system consists of a pair of 150 KW motor generators,
motor-generator control cabinets, and a power distribution cabinet. The
motor generators supply 208 V, 400 cycle, three-phase power to the power
distribution cabinet, which the power distribution cabinet supplies via a
variac to each power supply. The power distribution cabinet also contains
voltage and temperature monitoring equipment to detect power and cooling
malfunctions.

COOLING

Modules in the CRAY-1 computer system are cooled by the exchange of heat
from the module heat sink to a refrigerant-cooled cold bar. The module
heat sink is wedged along both 8-inch edges to a cold bar. Cold bars are
arranged in vertical columns, with each column having capacity for 144
modules. The cold bar is a cast aluminum bar containing a stainless steel
refrigerant tube.

MAINTENANCE CONTROL UNIT

The CRAY-1 computer system is equipped with a 16-bit minicomputer system
that serves as a maintenance tool and provides control for the system
initialization. After the CRAY-1 operating system has been initialized
and is operational, communication with the MCU is via a software protocol.
The MCU is connected to a CRAY-1 channel pair with additional control
signals for execution of the master clear operation, I/O master clear
operation, dead dump operation, and sample parity error operation.
The maintenance control unit (MCU) includes:

1. A Data General ECLIPSE minicomputer or equivalent with
    32K words of 16-bit memory
2. An 80-column card reader
3. A 132-column line printer
4. An 800 bpi 9-track tape unit
5. Two display terminals
6. A moving head disk drive

Included in the MCU system is a software package that enables it to
serve as a local batch station during production hours. As a local
station, diagnostic routines may be submitted for execution along with
other batch jobs. These diagnostics are typically stored on the local
disk and are submitted to the CRAY-1 by operator command.

The system initialization procedure is referred to in this manual as
the dead start sequence. This sequence is described in detail in
Section 3.

Detailed information about the MCU is presented in separate publications.

FRONT-END COMPUTER

The CRAY-1 computer system may be equipped with one or more front-end computer systems that provide input data to the CRAY-1 computer system and receive output from the CRAY-1 to be distributed to a variety of slow-speed peripheral equipments. A front-end computer system is a self-contained system that executes under the control of its own operating system. Peripheral equipment attached to the front-end computer will vary depending on the use to which the system is put.

A front-end computer may service the CRAY-1 in the following ways:
- As a local operator station
- As a local batch entry station
- As a data concentrator for multiplexing several other stations into a single CRAY-1 channel
- As a remote batch entry station

Detailed information about the front-end system is presented in separate publications.

EXTERNAL INTERFACE

The CRAY-1 may be interfaced to front-end systems through special interface controllers that compensate for differences in channel widths, machine word sizes, electrical logic levels, and control protocols. An interface is a Cray Research product and is contained in a small air-cooled stand-alone cabinet located near the front-end computer system. A primary goal of the interface is to maximize the utility of the front-end channel connected to the CRAY-1. Such a channel is generally slower than CRAY-1 channels. The CRAY-1 may be separated from the interface cabinet by up to 320 ft of cable with no degradation to its effective transfer rate. Maximum separation of the interface cabinet from the host processor is determined by the channel characteristics of the front-end machine. If site conditions require that the interconnected systems be physically located a considerable distance apart, the effective transmission rate may be degraded.

MASS STORAGE SUBSYSTEM

Mass storage for the CRAY-1 computer system consists of one or more Cray Research, Inc. DCU-2 Disk Controllers and multiple DD-19 Disk Storage Units. The disk controller is a Cray Research, Inc. product and is implemented in flat-pack ECL logic similar to that used in the CRAY-1 mainframe. The controller operates synchronously with the mainframe over a 16-bit full-duplex channel. The controller is in a DCC-1 refrigerant-cooled cabinet located near the mainframe. Up to four controllers may be contained in a cabinet. The cabinet requires about 5 sq. ft. of floor space and is 49 inches high.

Each controller may have from one to four DD-19 disk storage units attached to it. Data passes through the controller to or from one disk storage unit at a time. The controller may be connected to a 16-bit minicomputer station in addition to the CRAY-1. If this additional connection is made, the station and mainframe may share the controller operation. Either, but not both, can have an operation in progress at one time; software interlocks must be provided to avoid conflicts.

Each of the DD-19 disk storage units has two ports for controllers. A second independent data path may exist to each disk storage unit through another Cray Research controller. Reservation logic is provided to control access to each disk storage unit.

Operational characteristics of the DD-19 Disk Storage Units are summarized in Table 2-1. Further information about the mass storage subsystem is presented in separate publications.

Table 2-1. Characteristics of a DD-19 Disk Storage Unit

Bit capacity per drive	2.424×10^9	Latency	16.6 msec
Tracks per surface	411	Access time	15 - 80 msec
Sectors per track	18	Data transfer rate (average bits per sec.)	35.4×10^6
Bits per sector	32,768		
Number of head groups	10	Total bits that can be streamed to a unit (disk cylinder capacity)	5.9×10^6
Recording surfaces per drive	40		

SECTION 3
COMPUTATION SECTION
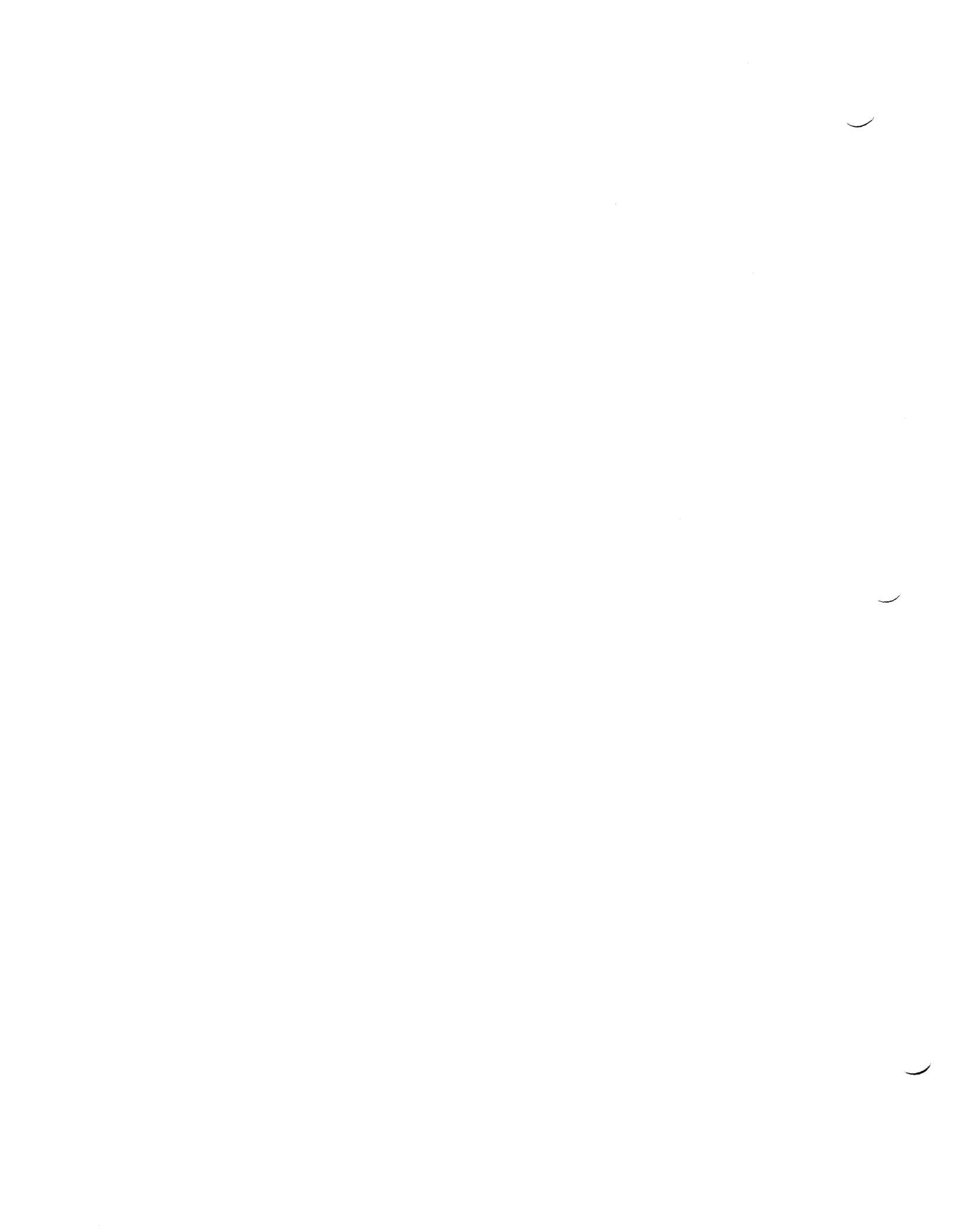
INTRODUCTION

The computation section (figure 3-1) consists of an instruction control
network, operating registers, and functional units. The instruction
control network performs all decisions related to instruction issue and
coordinates the activities for the three types of processing, vector,
scalar, and address. Associated with each type of processing are
registers and functional units that support the processing mode. For
vector processing, there are: a set of 64-bit 64-element registers,
three functional units dedicated solely to vector applications, and three
floating point functional units supporting both scalar and vector operations.
For scalar processing, there are two levels of 64-bit scalar registers and
four functional units dedicated solely to scalar processing in addition
to the three floating point units shared with the vector operations. For
address processing, there are two levels of 24-bit registers and two
integer arithmetic functional units.

Vector and scalar processing is performed on data as opposed to address
processing which operates on internal control information such as addresses
and indexes. The flow of data in the computation section is generally from
memory to registers and from registers to functional units. The flow of
results is from functional units to registers and from registers to memory
or back to functional units. Data flows along either the scalar or vector
path depending on the mode of processing it is undergoing. An exception is
that scalar registers can provide one of the operands required for vector
operations performed in the vector functional units.

The flow of address information is from memory or from control registers to
address registers. Information in the address registers can then be distributed
to various parts of the control network for use in controlling the scalar,
vector, and I/O operations. The address registers can also supply operands
to two integer functional units. The units generate address and index
information and return the result to the address registers. Address
information can also be transmitted to memory from the address registers.
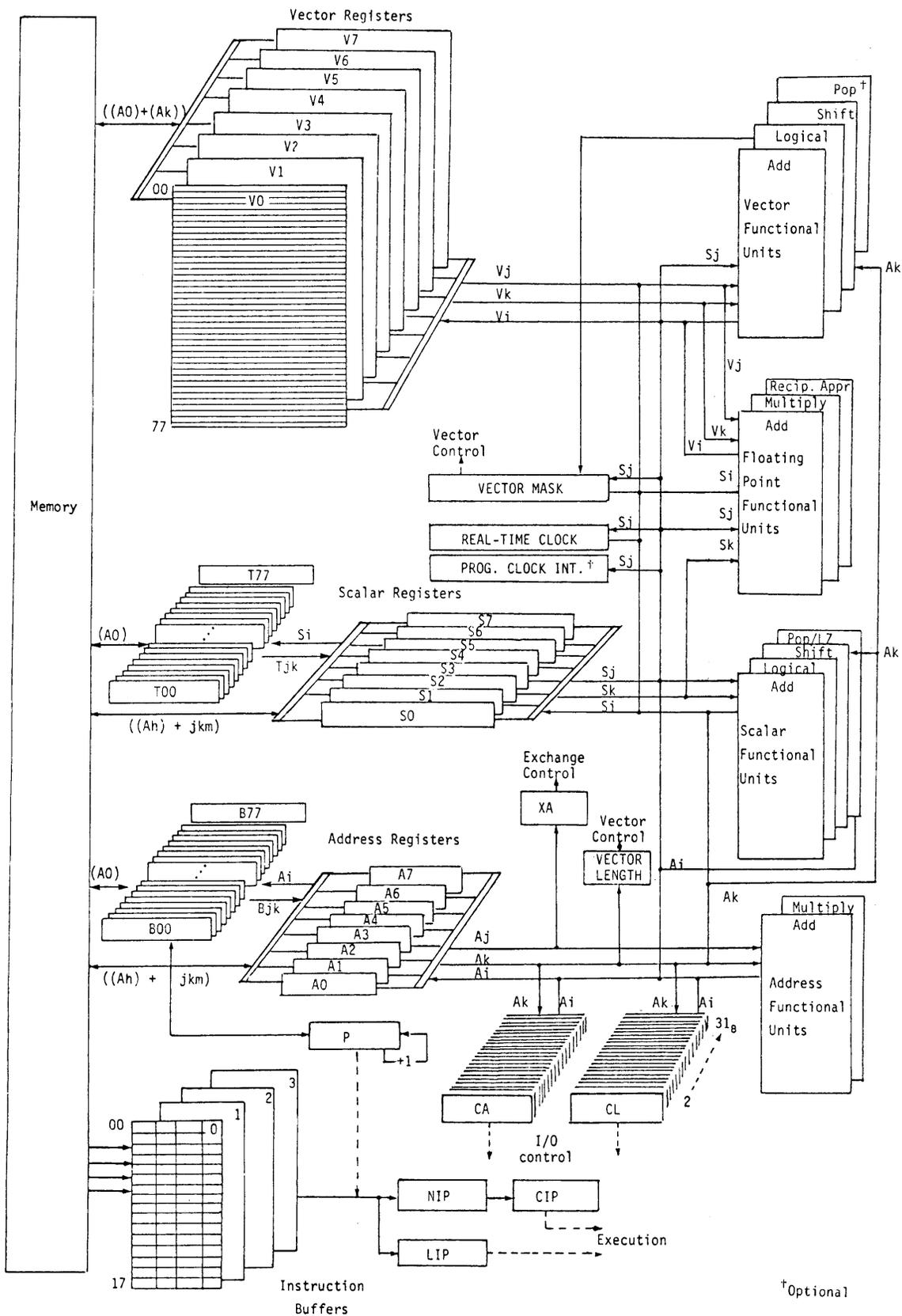
Figure 3-1. Computation section

REGISTER CONVENTIONS

Frequent use is made in this manual of parenthesized register names.
This is shorthand notation for the expression "the contents of register
---." For example, "Branch to (P) means "Branch to the address indicated
by the contents of the program parcel counter, P."

Extensive use is also made of subscripted designations for the A, B, S,
T, and V registers. For example, "Transmit (Tjk) to Si" means "Transmit
the contents of the T register specified by the jk designators to the S
register specified by the i designator."

In this manual, register bit positions are numbered from left to right
starting with bit 0. Bit 63 of an S, V, or T register value represents
the least significant bit in the operand. Bit 23 of an A or B register
value represents the least significant bit in the operand. When a power
of two is meant rather than a bit position, it is referred to as 2^n,
where n is the power of two.

OPERATING REGISTERS

Operating registers are a primary programmable resource of the CRAY-1.
They enhance the speed of the system by satisfying the heavy demands for
data that are made by the functional units. A single functional unit may
require one to three operands per clock period and may deliver results at
a rate of one per clock period. Moreover, multiple functional units can
be in use concurrently. To meet these requirements, the CRAY-1 has five
sets of registers; three primary sets and two intermediate sets. The
three primary sets of registers are vector, scalar, and address designated
in this manual as V, S, and A, respectively. These registers are considered
primary because functional units can access them directly. For the scalar
and address registers, an intermediate level of registers exists which is
not accessible to the functional units. These registers act as buffers
for the primary registers. Block transfers are possible between these
registers and memory so that the number of memory references required for
scalar and address operands is greatly reduced. The intermediate registers
that support scalar registers are referred to as T registers. The inter-
mediate registers that support the address registers are referred to as B
registers.

V REGISTERS

Eight V registers, each with 64 elements are the major computational registers of the CRAY-1. Each element of a V register has 64 bits. When associated data is grouped into successive elements of a V register, the register quantity may be considered a vector. Examples of vector quantities are rows or columns of a matrix or elements of a table.

Computational efficiency is achieved by processing each element of a vector identically. Vector instructions provide for the iterative processing of successive vector register elements. A vector operation begins by obtaining operands from the first element of one or more V registers and delivering the result to the first element of a V register. Successive elements are provided each clock period and as each operation is performed, the result is delivered to successive elements of the result V register. The vector operation continues until the number of operations performed by the instruction equals a count specified by the contents of the vector length (VL) register. Vectors having lengths exceeding 64 are handled under program control in groups of 64 and a remainder.

A result may be received by a V register and retransmitted as an operand to a subsequent operation in the same clock period. This use of a register as both a result and operand register allows for the "chaining" of two or more vector operations together. In this mode, two or more results may be produced per clock period.

The contents of a V register are transferred to or from memory in a block mode by specifying a first word address in memory, a positive or negative increment for computing memory addresses, and a vector length. The transfer then proceeds beginning with the first element of the V register at a maximum rate of one word per clock period, depending on bank conflicts.

Single-word data transfers are possible between an S register and an element of a V register.

In this manual, the V registers are individually referred to by the letter V and a numeric suffix in the range 0 through 7. Vector instructions reference V registers by allowing specification of the suffix as the i, j, or k designator as described in section 4 of this manual.

Individual elements of a V register are designated in this manual by decimal numbers in the range 00 through 63. These appear as subscripts to vector register references. For example, $V6_{29}$ refers to element 29 of vector register 6.

V register reservations

The term "reservation" describes the register condition when a register is in use and therefore not available for use as a result or as an operand register for another operation. During execution of a vector instruction, reservations are placed on the operand V registers and on the result V register. These reservations are placed on the registers themselves, not on individual elements of the V register.

A reservation for a result register is lifted during "chain slot" time. Chain slot time is the clock period that occurs at functional unit time plus two clock periods. During this clock period, the result is available for use as an operand in another vector operation. Chain slot time has no effect on the reservation placed on operand V registers.
A V register may serve only one vector operation as the source of one or both operands.
No reservation is placed on the VL register during vector processing. If a vector instruction employs an S register, no reservation is placed on the S register. It may be modified in the next instruction after vector issue without affecting the vector operation. The length and scalar operand (if appropriate) of each vector operation is maintained apart from the VL register. Vector operations employing different lengths may proceed concurrently; however, the vector length should not be changed between operations that chain because chaining implies operations of the same length.
The A_0 and Ak registers in a vector memory reference are treated in a similar fashion. They are available for modification immediately after use.

The vector store instruction (177) is blocked from chain slot execution.

The vector read instruction (176) is blocked from chain slot execution if the memory increment is a multiple of eight on a 16-bank machine or is a multiple of four on an 8-bank machine. A vector read cannot chain if speed control is in effect. Speed control is caused by bank conflicts due to the increment, which varies between 8 and 16 bank machines.

VECTOR CONTROL REGISTERS

Two registers are associated with vector registers and provide control information needed in the performance of vector operations. They are the vector length (VL) register and the vector mask (VM) register.

VL register

The 7-bit vector length register can be set to 0 through 100_8 and specifies the length of all vector operations performed by vector instructions and the length of the vectors held by the V registers. It controls the number of operations performed for instructions 140 through 177. The VL register may be set to an A register value through use of the 0020 instruction.

Cray Research cautions users against changing VL between operations that may chain together. In code sequences where the vector length is increased, unexpected results may occur.

Suppose, for example, that during a vector sequence the contents of VL are changed to a larger value and a second operation is initiated to chain to the first operation. The user may expect that the second operation will use the results of the first operation and the operands in the register unaltered by the first operation. However, when the instructions chain together, the second instruction does not receive the anticipated operands beyond the VL specified for the first operation. The user who intends to use the system in this manner must take care to avoid chained operations. Although there may be applications of the characteristic produced by chained operations with different contents for VL, Cray Research takes no responsibility for its use. Chained operation cannot be assured since I/O interrupts may "break" the chain.

VM register

The vector mask register has 64 bits, each of which corresponds to a word element in a vector register. Bit 0 corresponds to element 0, bit 63 to element 63. The mask is used in conjunction with vector merge and test instructions to allow operations to be performed on individual vector elements.

The vector mask register may be set from an S register through the 003 instruction or may be created by testing a vector register for condition using the 175 instruction. The mask controls element selection in the vector merge instructions (146 and 147).

E

S REGISTERS

The eight 64-bit S registers are the principal scalar registers for the CPU. These registers serve as the source and destination for operands in the execution of scalar arithmetic and logical instructions. The related functional units perform both integer and floating point arithmetic operations.

S registers may furnish one operand in vector instructions. Single-word transmissions of data between an S register and an element of a V register are also possible.

Data can move directly between memory and S registers or can be placed in T registers as an intermediate step. This allows buffering of scalar operands between S registers and memory.

Data can also be transferred between A and S registers.

Another use of the S registers is for setting or reading the vector mask (VM) register or the real-time clock register.

At most, one S register can be entered with data during each clock period. Issue of an instruction is delayed if it would cause data to arrive at the S registers at the same time as data already being processed which is scheduled to arrive from another source.

When an instruction issues that will deliver new data to an S register, a reservation is set for that register to prevent issue of instructions that read the register until the new data has been delivered.

In this manual, the S registers are individually referred to by the letter S and a numeric subscript in the range 0 through 7. Instructions reference S registers by allowing specification of the subscript as the i, j, or k designator as described in section 4 of this manual. The only register to which an implicit reference is made is the S_0 register. The use of this register is implied in the following branch instructions:

      014 through 017.

Refer to section 4 for additional information concerning the use of S registers by instructions.

T REGISTERS

There are sixty-four 64-bit T registers in the computation section. The T registers are used as intermediate storage for the S registers.

Data may be transferred bidirectionally between T and S registers and between T registers and memory. The transfer of a value between a T register and an S register requires only one clock period. T registers reference memory through block read and block write instructions. Block transfers occur at a maximum rate of one word per clock period. No reservations are made for T registers and no instructions can issue during block transfers to and from T registers.

In this manual, T registers are referred to by the letter T and a 2-digit octal subscript in the range 00 through 77. Instructions reference T registers by allowing specification of the octal subscript as the jk designator as described in section 4 of this manual.

A REGISTERS

The eight 24-bit A registers serve a variety of applications. They are primarily used as address registers for memory references and as index registers but also are used to provide values for shift counts, loop control, and channel I/O operations. In address applications, they are used to index the base address for scalar memory references and for providing both a base address and an index address for vector memory references.

The address functional units support address and index generation by performing 24-bit integer arithmetic on operands obtained from A registers and delivering the results to A registers.

Data can move directly between memory and A registers or can be placed in B registers as an intermediate step. This allows buffering of the data between A registers and memory.

Data can also be transferred between A and S registers.

The vector length register is set by transmitting a value to it from an A register.

At most, one A register can be entered with data during each clock period. Issue of an instruction is delayed if it would cause data to arrive at the A registers at the same time as data already being processed which is scheduled to arrive from another source.

When an instruction issues that will deliver new data to an A register, a reservation is set for that register to prevent issue of instructions that read the register until the new data has been delivered.

In this manual, the A registers are individually referred to by the letter A and a numeric subscript in the range 0 through 7. Instructions reference A registers by allowing specification of the subscript as the h, i, j, or k designator as described in section 4 of this manual. The only register to which an implicit reference is made is the A_0 register. The use of this register is implied in the following instructions:

    010 through 013
    034 through 037
    176 and 177

Refer to section 4 for additional information concerning the use of A registers by instructions.

B REGISTERS

There are sixty-four 24-bit B registers in the computation section. The B registers are used as intermediate storage for the A registers. Typically, the B registers will contain data to be referenced repeatedly over a sufficiently long span that it would not be desirable to retain the data in either A registers or in memory. Examples of uses are loop counts, variable array base addresses, and dimensions.

The transfer of a value between an A register and a B register requires only one clock period. A block of B registers may be transferred to or from memory at the maximum rate of one 24-bit value per clock period. No reservations are made for B registers and no instructions can issue during block transfers to and from B registers.

In this manual, B registers are individually referred to by the letter B and a 2-digit octal subscript in the range 00 through 77. Instructions reference B registers by allowing specification of the octal subscript as the jk designator as described in section 4 of this manual. The only B register to which an implicit reference is made is the B_{00} register. On execution of the return jump instruction (007), register B_{00} is set to the next instruction parcel address and a branch to an address specified by ijkm occurs. Upon receiving control, the called routine will conventionally save (B_{00}) so that the B_{00} register will be free for the called routine to initiate return jumps of its own. When a called routine wishes to return to its caller, it restores the saved address and executes a 005 instruction. This instruction, which is a branch to (Bjk), causes the address saved in Bjk to be entered into P as the address of the next instruction parcel to be executed.

FUNCTIONAL UNITS

Instructions other than simple transmits or control operations are performed by hardware organizations known as functional units. Each unit implements an algorithm or a portion of the instruction set. Units are independent; a number of functional units can be in operation at the same time.

A functional unit receives operands from registers and delivers the result to a register when the function has been performed. The units operate essentially in three-address mode with source and destination addressing limited to register designators.

All functional units perform their algorithms in a fixed amount of time; no delays are possible once the operands have been delivered to the unit. The amount of time required from delivery of the operands to the unit to the completion of the calculation is termed the "functional unit time" and is measured in 12.5 nsec clock periods.

The functional units are fully segmented. This means that a new set of operands for any computation may enter a functional unit each

clock period even though the functional unit time may be more than one clock period. This segmentation is made possible by capturing and holding the information arriving at the unit or moving within the unit at the end of every clock period.

Twelve functional units are identified in this manual and are arbitrarily described in four groups: address, scalar, vector, and floating point. The first three groups each act in conjunction with one of the three primary register types, A, S, and V, to support the address, scalar, and vector modes of processing available in the CRAY-1. The fourth group, floating point, can support either scalar or vector operations and will accept operands from or deliver results to S or V registers accordingly.

ADDRESS FUNCTIONAL UNITS

The address functional units perform 24-bit integer arithmetic on operands obtained from A registers and deliver the results to an A register. The arithmetic is two's complement.

Address add unit

The address add unit performs 24-bit integer addition and subtraction. The unit executes instructions 030 and 031. The addition and subtraction are performed in a similar manner. However, the two's complement subtraction for the 031 instruction occurs as follows. The one's complement of the Ak operand is added to the Aj operand. Then a one is added in the low order bit position of the result.

No overflow is detected in the functional unit.

The functional unit time is two clock periods.

Address multiply unit

The address multiply unit executes instruction 032, which forms a 24-bit integer product from two 24-bit operands. No rounding is performed. The result consists of the 24 least significant bits of the product.

The functional unit does not detect overflow of the product.

The function unit time is six clock periods.

SCALAR FUNCTIONAL UNITS

The scalar functional units perform operations on 64-bit operands obtained
from S registers and in most cases deliver the 64-bit results to an S
register. The exception is the population/leading zero count unit which
delivers its 7-bit result to an A register.

Four functional units are exclusively associated with scalar operations
and are described here. Three functional units are used for both scalar
and vector operations and are described under the section entitled
Floating Point Functional Units.

Scalar add unit

The scalar add unit performs 64-bit integer addition and subtraction. It
implements instructions 060 and 061. The addition and subtraction are per-
formed in a similar manner. However, the two's complement subtraction
for the 061 instruction occurs as follows. The one's complement of the Sk
operand is added to the Sj operand. Then a one is added in the low order
bit position of the result.

No overflow is detected in the unit.

The functional unit time is three clock periods.

Scalar shift unit

The scalar shift unit shifts the entire 64-bit contents of an S register
or shifts the double 128-bit contents of two concatenated S registers.
Shift counts are obtained from an A register or from the jk portion of
the instruction. Shifts are end off with zero fill. For a double shift,
a circular shift is effected if the shift count does not exceed 64 and
the i and j designators are equal and non-zero.

The scalar shift unit implements instructions 052 through 057. Single
register shift instructions, 052 through 055, are executed in two clock
periods. Double-register shift instructions, 056 and 057, are executed
in three clock periods.

Scalar logical unit

The scalar logical unit performs bit-by-bit manipulation of 64-bit quantities obtained from S registers. It implements instructions 042 through 051, the mask and Boolean instructions. An operation requires only one clock period.

Population/leading zero count unit

This functional unit implements instructions 026 and 027. The 026 instruction, which counts the number of bits having a value of one in the operand, executes in four clock periods. The 027 instruction, which counts the number of bits of zero preceding a one bit in the operand, executes in three clock periods. For either instruction, the 64-bit operand is obtained from an S register and the 7-bit result is delivered to an A register.

When the Vector Population Instructions Option is installed, this unit also recognizes an additional instruction, the 026ij1 instruction, which returns a one-bit population count parity (even) of an S register's contents to an A register.

VECTOR FUNCTIONAL UNITS

Most vector functional units perform operations on operands obtained from one or two V registers or from a V register and an S register. The reciprocal unit, which requires only one operand, is an exception. Results from a vector functional unit are delivered to a V register.

Successive operand pairs are transmitted to a functional unit each clock period. The corresponding result emerges from the functional unit n clock periods later where n is the functional unit time and is constant for a given functional unit. The vector length determines the number of operand pairs to be processed by a functional unit.

Three functional units are exclusively associated with vector operations
and are described in this subsection. Three functional units are associated
with both vector operations and scalar operations and are described in the
subsection entitled Floating Point Functional Units. When a floating point
unit is used for a vector operation, the general description of vector
functional units given in this subsection applies.

Vector functional unit reservation

A functional unit engaged in a vector operation remains busy during each
clock period and may not participate in other operations. In this state,
the functional unit is said to be reserved. Other instructions that
require the same functional unit will not issue until the previous
operation is completed. Only one functional unit of each type is
available to the vector instruction hardware. When the vector operation
completes, the reservation is dropped and the functional unit is then
available for another operation.

Recursive characteristic of vector functional units

In a vector operation, the result register (designated by i in the
instruction) is not normally the same V register as the source of either
of the operands (designated by j or k). However, turning the output
stream of a vector functional unit back into the input stream by setting
i to the same register designator as j or k may be desirable under certain
circumstances since it provides a facility for reducing 64 elements down
to just a few. The number of terms generated by the partial reduction is
determined by the number of values that can be in process in a functional
unit at one time (i.e., functional unit time + 2CP).

When the i designator is the same as the j or k designator, a recursive
characteristic is introduced into the vector processing because of the
way in which element counters are handled. At the beginning of an operation
for which i is the same as j or k, the element counters for both the operand
register and the operand/result register are set to zero. The element
counter for the operand/result register is held at zero and does not begin

incrementing until the first result arrives from the functional unit at functional unit time + 2 CP. This counter then begins to advance by one each clock period. Note that until f.u. + 2, the initial contents of element zero of the operand/result register are repeatedly sent to the functional unit. The element counter for the other operand register, however, immediately begins advancing by one on each successive clock period thus sending the contents of elements 0, 1, 2, ... on successive clock periods. Thus, the first f.u. + 2 elements of the operand/result register contain results based on the contents of element 0 of the operand/result register and on successive elements of the other operand register. These f.u. + 2 elements then provide one of the operands used in calculating the results for the next f.u. + 2 elements. The third group of f.u. + 2 elements of the operand/result register contains results based on the results delivered to the second group of f.u. + 2 elements, and so on until the final group of f.u. + 2 elements is generated as determined by the vector length.

As an example, consider the summation of a vector of floating point numbers where the initial conditions for the vector operation are the following:
- All elements of register V1 contain floating point values.
- Register V2 will provide one set of operands and will receive the results. Element 0 of this register contains a 0 value.
- The vector length register (VL) contains 64.

A floating point add instruction (171212) is then executed using register V1 for one operand and using register V2 as an operand/result register. This instruction uses the floating point add unit which has a functional unit time of 6 CP causing sums to be generated in groups of eight (f.u. + 2 = 8). The final eight partial sums of the 64 elements of V1 are contained in elements 56 through 63 of V2. Specifically, elements of V2 contain the following sums:

$$(V2_{00}) = (V2_{00}) + (V1_{00})$$
$$(V2_{01}) = (V2_{00}) + (V1_{01})$$
$$(V2_{02}) = (V2_{00}) + (V1_{02})$$
$$(V2_{03}) = (V2_{00}) + (V1_{03})$$
$$(V2_{04}) = (V2_{00}) + (V1_{04})$$

(contents of register V2, continued)

$$(V2_{05}) = (V2_{00}) + (V1_{05})$$
$$(V2_{06}) = (V2_{00}) + (V1_{06})$$
$$(V2_{07}) = (V2_{00}) + (V1_{07})$$

$$\overbrace{\text{new } (V2_{00})}$$

$$(V2_{08}) = (V2_{00}) + (V1_{08}) = (V2_{00}) + (V1_{00}) + (V1_{08})$$
$$(V2_{09}) = (V2_{01}) + (V1_{09}) = (V2_{00}) + (V1_{01}) + (V1_{09})$$
$$(V2_{10}) = (V2_{02}) + (V1_{10}) = (V2_{00}) + (V1_{02}) + (V1_{10})$$
$$(V2_{11}) = (V2_{03}) + (V1_{11}) = (V2_{00}) + (V1_{03}) + (V1_{11})$$
$$(V2_{12}) = (V2_{04}) + (V1_{12}) = (V2_{00}) + (V1_{04}) + (V1_{12})$$
$$(V2_{13}) = (V2_{05}) + (V1_{13}) = (V2_{00}) + (V1_{05}) + (V1_{13})$$
$$(V2_{14}) = (V2_{06}) + (V1_{14}) = (V2_{00}) + (V1_{06}) + (V1_{14})$$
$$(V2_{15}) = (V2_{07}) + (V1_{15}) = (V2_{00}) + (V1_{07}) + (V1_{15})$$
$$(V2_{16}) = (V2_{08}) + (V1_{16}) = (V2_{00}) + (V1_{00}) + (V1_{08}) + (V1_{16})$$

$$\vdots$$

$$(V2_{56}) = (V2_{48}) + (V1_{56}) = (V2_{00}) + (V1_{00}) + (V1_{08}) + (V1_{16}) \ldots + (V1_{56})$$
$$(V2_{57}) = (V2_{49}) + (V1_{57}) = (V2_{00}) + (V1_{01}) + (V1_{09}) + (V1_{17}) \ldots + (V1_{57})$$
$$(V2_{58}) = (V2_{50}) + (V1_{58}) = (V2_{00}) + (V1_{02}) + (V1_{10}) + (V1_{18}) \ldots + (V1_{58})$$
$$(V2_{59}) = (V2_{51}) + (V1_{59}) = (V2_{00}) + (V1_{03}) + (V1_{11}) + (V1_{19}) \ldots + (V1_{59})$$
$$(V2_{60}) = (V2_{52}) + (V1_{60}) = (V2_{00}) + (V1_{04}) + (V1_{12}) + (V1_{20}) \ldots + (V1_{60})$$
$$(V2_{61}) = (V2_{53}) + (V1_{61}) = (V2_{00}) + (V1_{05}) + (V1_{13}) + (V1_{21}) \ldots + (V1_{61})$$
$$(V2_{62}) = (V2_{54}) + (V1_{62}) = (V2_{00}) + (V1_{06}) + (V1_{14}) + (V1_{22}) \ldots + (V1_{62})$$
$$(V2_{63}) = (V2_{55}) + (V1_{63}) = (V2_{00}) + (V1_{07}) + (V1_{15}) + (V1_{23}) \ldots + (V1_{63})$$

Note that if an integer summation were performed instead of a floating
point summation, five partial sums would be generated and placed in
elements 59 through 63 since the functional unit time for the integer add
unit is 3 CP. Assuming that the same registers are used as for the previous
example but that the registers now contain integer values, the last five
elements of V2 would contain the following values:

$$(V2_{59}) = (V2_{00}) + (V1_{04}) + (V1_{09}) + (V1_{14}) \ldots + (V1_{59})$$
$$(V2_{60}) = (V2_{00}) + (V1_{00}) + (V1_{05}) + (V1_{10}) \ldots + (V1_{55}) + (V1_{60})$$
$$(V2_{61}) = (V2_{00}) + (V1_{01}) + (V1_{06}) + (V1_{11}) \ldots + (V1_{56}) + (V1_{61})$$
$$(V2_{62}) = (V2_{00}) + (V1_{02}) + (V1_{07}) + (V1_{12}) \ldots + (V1_{57}) + (V1_{62})$$
$$(V2_{63}) = (V2_{00}) + (V1_{03}) + (V1_{08}) + (V1_{13}) \ldots + (V1_{58}) + (V1_{63})$$

This recursive characteristic of vector processing is applicable to any
vector operation, arithmetic or logical. The value initially placed in
element 0 of the operand/result register will depend on the operation
being performed. For example, when using the floating point multiply
unit, element 0 of the operand/result register will usually be set to an
initial value of 1.0.

Vector add unit

The vector add unit performs 64-bit integer addition and subtraction for
a vector operation and delivers the results to elements of a V register.
The unit implements instructions 154 through 157. The addition and sub-
traction are performed in a similar manner. However, for the subtraction
operations, 156 and 157, the Vk operand is complemented prior to addition
and during the addition a one is added into the low order bit position of
the result.
No overflow is detected by the unit.

The functional unit time for the vector add unit is three clock periods.

Vector shift unit

The vector shift unit shifts the entire 64-bit contents of a V register
element or the 128-bit value formed from two consecutive elements of a
V register. Shift counts are obtained from an A register. Shifts are
end-off with zero fill.

The vector shift unit implements instructions 150 through 153. Functional
unit time is four clock periods.

Vector logical unit

The vector logical unit performs bit-by-bit manipulation of 64-bit
quantities for instructions 140 through 147. The unit also performs the
logical operations associated with the vector mask instruction, 175.
Because the 175 instruction uses the same functional unit as instructions
140 through 147, it cannot be chained with these logical operations.

Functional unit time is two clock periods.

Vector population count unit

Although the CRAY-1 does not include a vector population unit as a standard feature, such a unit is present when the Vector Population Instructions Option is installed. The vector population count unit recognizes the vector population count instruction, 174ij1 and the vector population count parity instruction, 174ij2. Because implementation of these instructions requires modifications to the format of the vector reciprocal approximation instruction, some of the restrictions for the reciprocal approximation unit hold true for the vector population instructions.

FLOATING POINT FUNCTIONAL UNITS

The three floating point functional units perform floating point arithmetic for both scalar and vector operations. When executing a scalar instruction, operands are obtained from S registers and the result is delivered to an S register. When executing most vector instructions, operands are obtained from pairs of V registers or from a V register and an S register and the results are delivered to a V register. The reciprocal instruction, which has only one input operand, is an exception.

A floating point unit is reserved during execution of a vector instruction.

Information on floating point out-of-range conditions is contained in the subsection entitled Floating Point Arithmetic.

Floating point add unit

The floating point add unit performs addition or subtraction of 64-bit operands in floating point format. The unit implements instructions 062, 063, and 170 through 173. Functional unit time is six clock periods.

A result is normalized even if the operands are unnormalized.

Out-of-range exponents are detected as described under Floating Point Arithmetic.

Floating point multiply unit

The floating point multiply unit executes instructions 064 through 067 and 160 through 167. These instructions provide for full and half precision multiplication of 64-bit operands in floating point format and for computing two minus a floating point product for reciprocal iterations.

The half-precision product is rounded; the full-precision product is either rounded or unrounded.

Input operands are assumed to be normalized. The unit delivers a normalized result except that the result is not guaranteed to be correct if the input operands are not normalized.

Out-of-range exponents are detected as described under Floating Point Arithmetic. However, if both operands have zero exponents, the result is considered as an integer product and is not normalized.

Functional unit time is seven clock periods.

Reciprocal approximation unit

The reciprocal approximation unit finds the approximate reciprocal of a 64-bit operand in floating point format. The unit executes instructions 070 and 174. If the Vector Population Instructions Option is installed, the k field must be 0 for the reciprocal approximation instruction, 174, to be recognized. Functional unit time is 14 clock periods.

The result is normalized. The input operand is assumed to be normalized; the uppermost bit of the coefficient is not tested but is assumed to be set in the computation.

ARITHMETIC OPERATIONS

Functional units in the CRAY-1 either perform two's complement integer
arithmetic or perform floating point arithmetic.

INTEGER ARITHMETIC

All integer arithmetic, whether 24 bits or 64 bits, is two's complement
and is so represented in the registers as illustrated in figure 3-2.
The address add unit and address multiply unit perform 24-bit arithmetic.
The scalar add unit and the vector add unit perform 64-bit arithmetic.

```
O                        23
┌─┬──────────────────────────┐
│ │                          │
└─┴──────────────────────────┘
SIGN
```
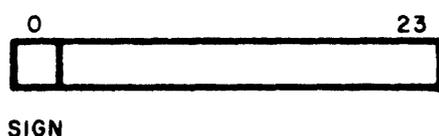
2's COMPLEMENT INTEGER (24 BITS)

```
O                                              63
┌─┬──────────────────────────────────────────────┐
│ │                                              │
└─┴──────────────────────────────────────────────┘
SIGN
```
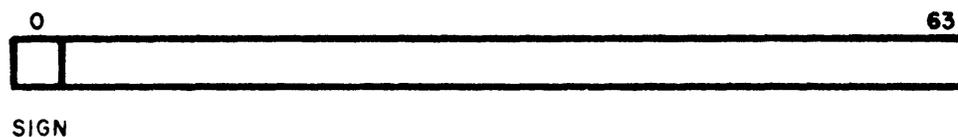
2's COMPLEMENT INTEGER (64 BITS)

Figure 3-2. Integer data formats

Multiplication of two integer operands may be accomplished using the
floating point multiply instruction. The floating point multiply unit
recognizes the conditions where both operands have zero exponents as a
special case and returns the upper 48 bits of the product of the
coefficients as the coefficient of the result and leaves the exponent
field zero.

Division of integers would require that they first be converted to
floating point format and then divided using the floating point units.

FLOATING POINT ARITHMETIC

Floating point numbers are represented in a standard format throughout the CPU. This format is a packed representation of a binary coefficient and an exponent or power of two. The coefficient is a 48-bit signed fraction. The sign of the coefficient is separated from the rest of the coefficient as shown in figure 3-3. Since the coefficient is signed magnitude, it is not complemented for negative values.

BINARY POINT

```
O  I              15  16                                    63
┌─┬──────────────┬───────────────────────────────────────┐
│ │              │                                         │
└─┴──────────────┴───────────────────────────────────────┘
SIGN   EXPONENT                    COEFFICIENT
```
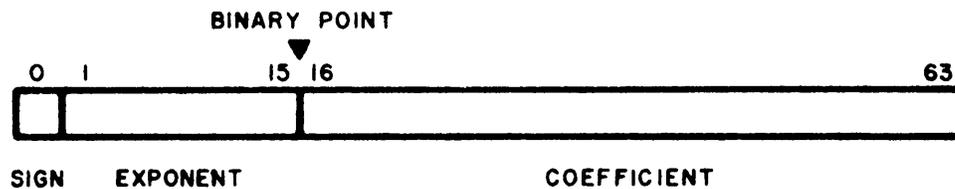
Figure 3-3. Floating point data format

The exponent portion of the floating point format is represented as a biased integer in bits 1 through 15. The bias that is added to the exponents is 40000_8. The positive range of exponents is 40000_8 through 57777_8. The negative range of exponents is 37777_8 through 20000_8. Thus, the unbiased range of exponents is the following:

$$2^{-20000}{}_8 \text{ through } 2^{+17777}{}_8$$

In terms of decimal values, the floating point format of the CRAY-1 allows the expression of numbers accurate to about 15 decimal digits in the approximate decimal range of 10^{-2466} through 10^{+2466}.

A zero value or an underflow result is not biased and is represented as a word of all zeros.

A negative zero is not generated by any functional unit.

Normalized floating point

A non-zero floating point number in packed format is normalized if the most significant bit of the coefficient is non-zero. This condition implies that the coefficient has been shifted to the left as far as possible and therefore the floating point number has no leading zeros in the coefficient.

When a floating point number has been created by inserting an exponent of 40060_8 into a word containing a 48-bit integer, the result should be normalized before being used in a floating point operation. Normalization is accomplished by adding the unnormalized floating point operand to zero. Since S_0 provides a 64-bit zero when used in the Sj field of an instruction, a normalize of an operand in Sk can be performed using the following instruction:

        062i0k

Si contains the normalized result.

Floating point range errors

Overflow of the floating point range is indicated by an exponent value of 60000_8 or greater in packed format. Underflow is indicated by an exponent value of 17777_8 or less in packed format. Detection of the overflow condition will initiate an interrupt if the floating point mode flag is set in the mode register and monitor mode one is not in effect. The floating point mode flag can be set or cleared by an object program.

Detection of floating point range error conditions by the floating point units is described in the following paragraphs.

<u>Floating point add unit</u> - A floating point add range error condition is generated for scalar operands when the larger incoming exponent is greater than or equal to 60000_8. The floating point error flag is set and an exponent of 60000_8 is sent to the result register along with the computed coefficient, as in the following example:

```
  60000.4  Range error
+ 57777.4
  ─────────
  60000.6  Result register.
```

<u>Floating point multiply unit</u> - In the floating point multiply unit, if the exponent of either operand is greater than or equal to 60000_8 or if the sum of the two exponents is greater than or equal to 60000_8, the floating point error flag is set and an exponent of 60000_8 is sent to the result register along with the computed coefficient.

An underflow condition is detected when the sum of the exponents is less than or equal to 17777_8 and causes an all zero exponent and coefficient to be returned to the result register. However, if the sum of the exponents is 20000_8 and a normalizing left shift occurs, an exponent of 17777_8 is sent to the result register along with the computed coefficient.

Underflow is also generated when either, but not both, of the incoming exponents is zero. Both exponents equal to zero is treated as an integer multiply and the result is treated normally with no normalization shift of the result allowed. The result is a 48-bit quantity starting with bit 16. When using this feature, consider the operands as 24-bit integers in bits 16 through 39 even though they are actually fractions with the binary point between bits 15 and 16. In the following example, operand 1 is 4 and operand 2 is 5 to produce a 48-bit result of 24.

```
                    0               16                  40              63
Operand 1        | 0 --- 0        |                    4|////////////////|

Operand 2        | 0 --- 0  .     |                    5|////////////////|

Result           | 0 --- 0        |                                    24|
```
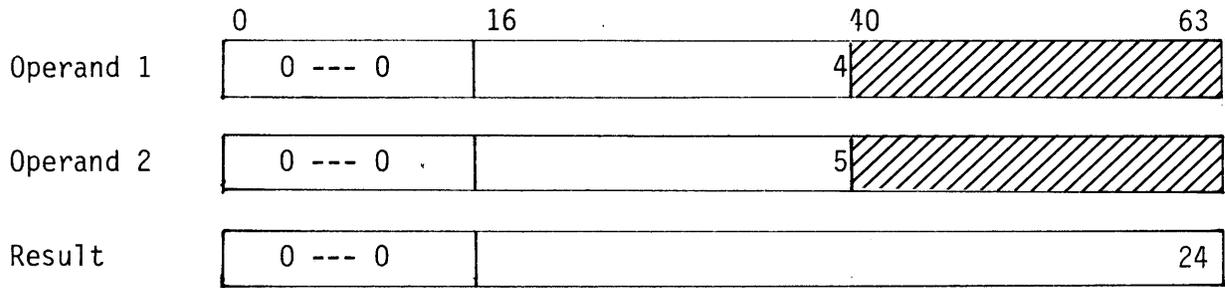
Floating point reciprocal approximation unit - For the floating point
reciprocal approximation unit, an incoming operand with an exponent less
than or equal to 20001_8 or greater than or equal to 60000_8 causes a
floating point range error. The error flag is set and an exponent of
60000_8 is sent to the result register along with the computed coefficient.

Double precision numbers

The CRAY-1 does not provide special hardware for performing double or
multiple precision operations. Double precision computations with 95-bit
accuracy are available through software routines provided by Cray Research.

Addition algorithm

Floating point addition or subtraction is performed in a 49-bit register.
Trial subtraction of the exponents occurs to select the operand to be
shifted down for aligning the operands. The larger exponent operand
carries the sign and the shift is always to the right. Bits shifted
out of the register are lost; no round-up takes place.

```
   |1| . |              48              |
                                                    discarded
                                                       ↙
            •        |             |////////////////|
```
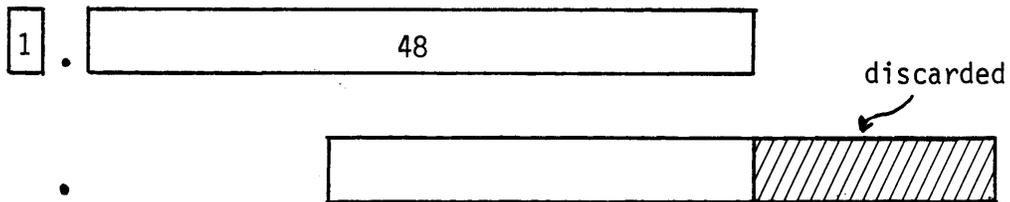
Figure 3-4. 49-bit floating point addition

Multiplication algorithm

The floating-point multiply unit in the CPU has an input of 48 bits of coefficient into a multiply pyramid (figure 3-5). The pyramid truncates part of the lower bits of the 96-bit product. To adjust for this truncation, a constant is unconditionally added above the truncation. The value determined by summing all carries produced by all possible combinations that could be truncated, and dividing the sum by the number of possible combinations. This averages to nine carries which are injected at the 2^{-56} position.

The errors due to this truncation and rounding are in the range:

$$-0.23 \times 2^{-48} \text{ to } +0.57 \times 2^{-48}$$
$$\text{or } -8.17 \times 10^{-16} \text{ to } +20.25 \times 10^{-16}.$$

The effect of this error is at most a round up of bit 2^{-48} of the result.
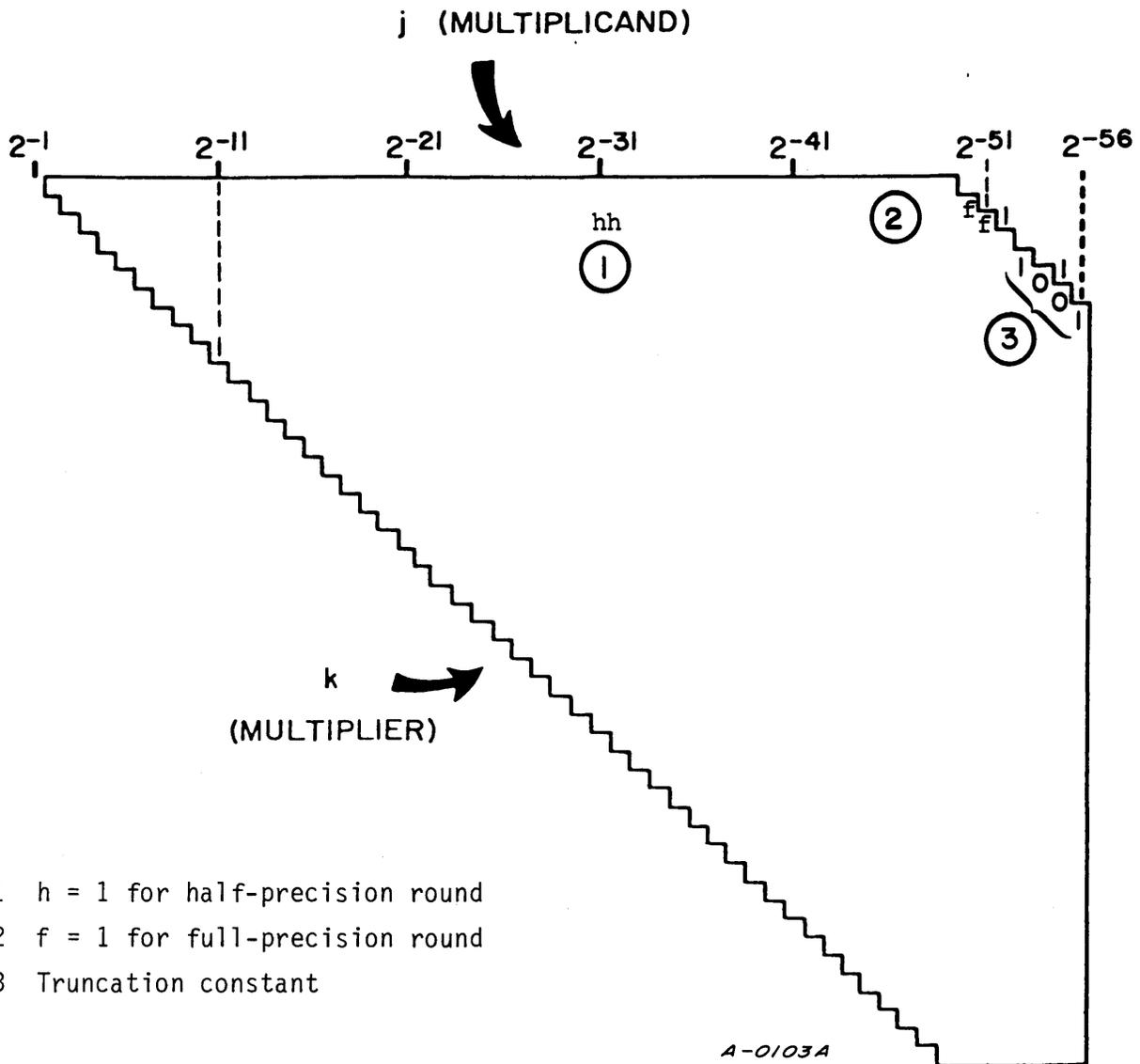
Figure 3-5. F.P. multiply partial product sums pyramid

The multiplication is commutative, that is, A times B equals B times A.

In a full-precision rounded multiply, 2 round bits are entered into the pyramid at bit position 2^{-50} and 2^{-51} and allowed to propagate up the pyramid.

For a half precision multiply, round bits are entered into the pyramid at bit positions 2^{-32} and 2^{-31}. A carry resulting from this entry is allowed to propagate up and a 30-bit result (2^{-1} to 2^{-30}) is transmitted back.

Division algorithm

The CRAY-1 performs floating point division by the method of reciprocal approximation. This facilitates the hardware implementation of a fully-segmented functional unit. Operands may enter the reciprocal unit each clock period because of this segmentation. In vector mode, results are produced at a one clock period rate. These results may be used in other vector operations during chaining because all functional units in the CRAY-1 have the same result rate.

The division algorithm that computes S_1 / S_2 to full precision requires four operations:

1. $S_3 = 1/S_2$ Reciprocal approximation
2. $S_4 = (2 - S_3 * S_2)$ Reciprocal iteration
3. $S_5 = S_1 * S_3$ Numerator * approximation
4. $S_6 = S_4 * S_5$ Half-precision quotient * correction factor

The approximation is based on Newton's method. The reciprocal approximation at step 1 is correct to 30 bits. The additional Newton iteration at step 2 increases this accuracy to 47 bits. This iteration is applied as a correction factor with a full-precision multiply operation.

Where 31 bits of accuracy is sufficient, the reciprocal approximation instruction may be used with the half-precision multiply to produce a half-precision quotient.

The 18 low-order bits of the half-precision results are returned as zeros with a round applied to the low-order bit of the 30-bit result.

A scalar quotient is computed in 29 clock periods since operations 2 and 3 issue in successive clock periods.

A vector quotient requires effectively three vector times since operations 1 and 3 are chained together. This hides one of the multiply operations. A vector time is one clock period for each element in the vector.

For example, two 50-element vectors are divided in about 3 * 50 clock periods. This estimate does not include overhead associated with the functional units.

LOGICAL OPERATIONS

The scalar and vector logical units perform bit-by-bit manipulation
of 64-bit quantities. Operations provide for forming logical products,
differences, sums and merges.

A logical product is the AND function:

operand one	1 0 1 0
operand two	<u>1 1 0 0</u>
result	1 0 0 0

A logical difference is the exclusive OR function:

operand one	1 0 1 0
operand two	<u>1 1 0 0</u>
result	0 1 1 0

A logical sum is the inclusive OR function:

operand one	1 0 1 0
operand two	<u>1 1 0 0</u>
result	1 1 1 0

INSTRUCTION ISSUE AND CONTROL

This section describes the instruction buffers and registers involved with instruction issue and control. Figure 3-6 illustrates the general flow of instruction parcels through the registers and buffers.
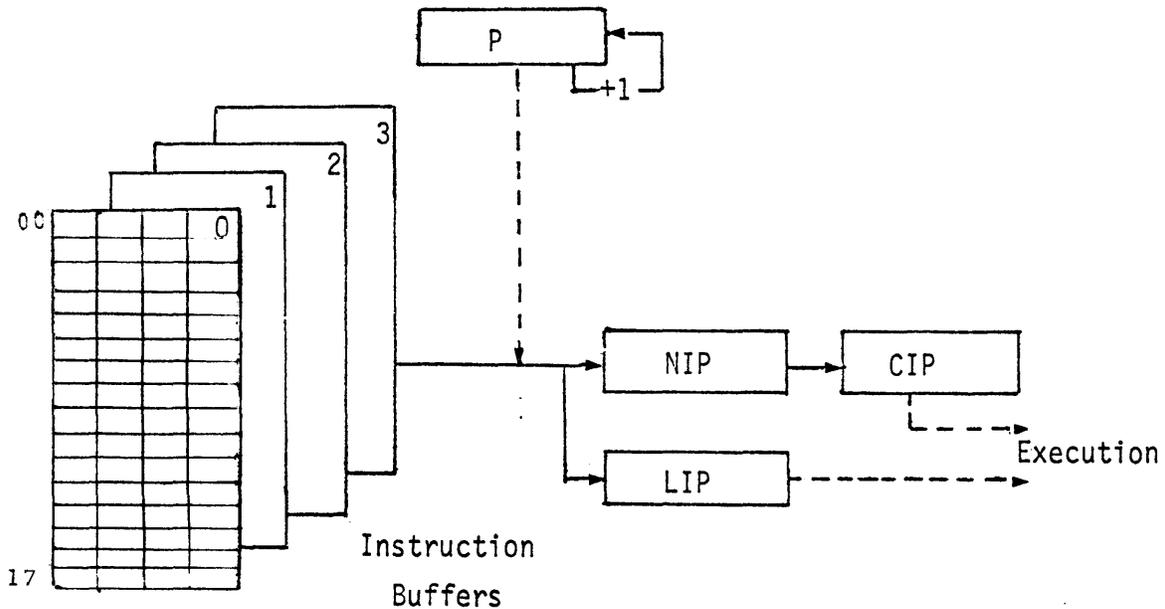
Figure 3-6.   Relationship of instruction buffers
and registers

P REGISTER

The P register is a 22-bit register which indicates the next parcel of program code to enter the next instruction parcel (NIP) register in a linear program sequence. The upper 20 bits of the P register indicate the word address for the program word in memory. The lower two bits indicate the parcel within the word. The content of the P register is normally advanced as each parcel successfully enters the NIP register. The value in the P register normally corresponds to the parcel address for the parcel currently moving to the NIP register.

The P register is entered with new data on an instruction branch or
on an exchange sequence. It is then advanced sequentially until the
next branch or exchange sequence. The value in the P register is
stored directly into the terminating exchange package during an
exchange sequence.

The P register is not master cleared. An undetermined value is stored in
the terminating exchange package at address zero during the dead start
sequence.

CIP REGISTER
The CIP (current instruction parcel) register is a 16-bit register
which holds the instruction waiting to issue. If this instruction
is a two-parcel instruction, the CIP register holds the upper half
of the instruction and the LIP holds the lower half. Once an
instruction enters the CIP register, it must issue. Issue may be
delayed until previous operations have been completed but then the
current instruction waiting for issue must proceed. Data arrives
at the CIP register from the NIP register. The indicators which make
up the instruction are distributed to all modules which have mode
selection requirements when the instruction issues.

The control flags associated with the CIP register are generally master
cleared. The register itself is not and an undetermined instruction will
issue during the master clear sequence.

NIP REGISTER
The NIP (next instruction parcel) register is a 16-bit register
which holds a parcel of program code prior to entering the CIP
register. A parcel of program code which has entered the NIP
register must be executed. There is no mechanism to discard it.

The NIP register is not master cleared. An undetermined instruction may issue during the master clear interval before the interrupt condition blocks data entry into the NIP register.

LIP REGISTER

The LIP (lower instruction parcel) register is a 16-bit register which holds the lower half of a two-parcel instruction at the time the two-parcel instruction issues from the CIP register.

INSTRUCTION BUFFERS

There are four instruction buffers in the CRAY-1, each of which holds 64 consecutive 16-bit instruction parcels (figure 3-7). Instruction parcels are held in the buffers prior to being delivered to the NIP or LIP registers.

Bank				
0	0	1	2	3
1	4	5	6	7
2	10	11	12	13
3	14	15	16	17
4	20	21	22	23
5	24	25	26	27
6	30	31	32	33
7	34	35	36	37
10_8	40	41	42	43
11_8	44	45	46	47
12_8	50	51	52	53
13_8	54	55	56	57
14_8	60	61	62	63
15_8	64	65	66	67
16_8	70	71	72	73
17_8	74	75	76	77

Buffer 3
Buffer 2
Buffer 1
Buffer 0
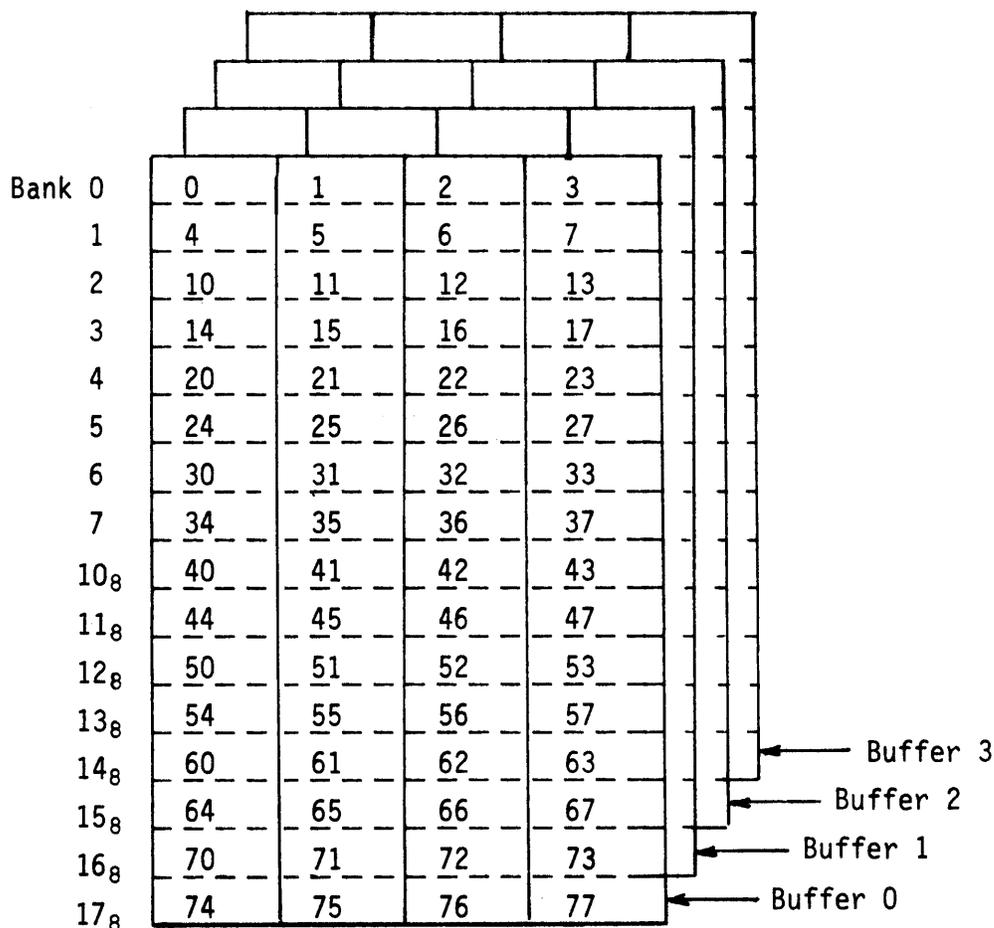
Figure 3-7 Instruction buffers

The beginning instruction parcel in a buffer always has a parcel address
that is an even multiple of 100_8. This allows the entire range of
addresses for instructions in a buffer to be defined by the high-order 16
bits of the beginning parcel address. For each buffer, there is a 16-bit
beginning address register that contains this value.

The beginning address registers are scanned each clock period. If the
high-order 18 bits of the P register match one of the beginning addresses,
an in-buffer condition exists and the proper instruction parcel is
selected from the instruction buffer. An instruction parcel to be
executed is normally sent to the NIP. However, the second half of a
two-parcel instruction is blocked from entering the NIP and is sent to
the LIP, instead, and is available when the upper half issues from the
CIP. At the same time, a blank parcel is entered into the NIP.

On an in-buffer condition, if the instruction is in a different buffer
than the previous instruction, a change of buffers occurs necessitating a
two clock period delay of issue.

An out-of-buffer condition exists when the high-order 18 bits of the P
register do not match any instruction buffer beginning address. When
this condition occurs, instructions must be loaded into one of the
instruction buffers from memory before execution can continue. The
instruction buffer that receives the instructions is determined by a two-
bit counter. Each occurrence of an out-of-buffer condition causes the
counter to be incremented by one so that the buffers are selected in
rotation.

Buffers are loaded from memory four words per clock period, an operation
that fully occupies memory. The first group of 16 parcels delivered to
the buffer always contains the instruction required for execution. For
this reason, the branch out of buffer time is a constant 14 clock periods.[†]
The remaining groups arrive at a rate of 16 parcels per clock period and
circularly fill the buffer.

---

† Refer to 8 Bank Phasing Option, section 5.

An instruction buffer is loaded with one word of instructions from each
of the 16 memory banks.† The first four instruction parcels residing in
an instruction buffer are always from bank 0. Figure 3-7 illustrates
the organization of parcels and words in an instruction buffer.

An exchange sequence voids the instruction buffers by setting their
beginning address registers to all ones. This prevents a match with the
P register and causes one of the buffers to be loaded.

Both forward and backward branching is possible within the buffers. A
branch does not cause reloading of an instruction buffer if the instruc-
tion being branched to is within one of the buffers. Multiple copies of
instruction parcels cannot occur in the instruction buffers. Because
instructions are held in instruction buffers prior to issue, no attempt
should be made to dynamically modify instruction sequences. As long as
the unmodified instruction is in an instruction buffer, the modified
instruction in memory will not be loaded into an instruction buffer.

Although optimization of code segment lengths for instruction buffers is
not a prime consideration when programming the CRAY-1, the number and
size of the buffers and the capability for both forward and backward
branching can be used to good advantage. Large loops containing up to 256
consecutive instruction parcels can be maintained in the four buffers or as
an alternative, one could have a main program sequence in one or two of the
buffers which makes repeated calls to short subroutines maintained in the
other buffers. The program and subroutines remain in the buffers undisturbed
as long as no out-of-buffer condition causes a buffer to be reloaded.

----

† Refer to 8-bank phasing option, section 5.

EXCHANGE MECHANISM

Exchange mechanism refers to the technique employed in the CRAY-1 for switching instruction execution from program to program. This technique involves the use of blocks or program parameters known as exchange packages and a CPU operation referred to as an exchange sequence. Three special registers are instrumental in the exchange mechanism. These are the exchange address (XA) register, the mode (M) register, and the flag (F) register.

XA REGISTER

The XA (exchange address) register specifies the first word address of a 16-word exchange package loaded by an exchange operation. The register contains the upper eight bits of a 12-bit field that specifies the address. The lower bits of the field are always zero; an exchange package must begin on a 16-word boundary. The 12-bit limit requires that the absolute address be in the lower 4096 words of memory.

When an execution interval terminates, the exchange sequence exchanges the contents of the registers with the contents of the exchange package at (XA)*16 in memory.

M REGISTER

The M (mode) register is a five-bit register that contains part of the exchange package for a currently active program. The five bits are selectively set during an exchange sequence. Bits are assigned in words n+1 and n+2 of the exchange package, figure 3-8, as follows:

n+1 Bit 39 Interrupt monitor mode select. This bit is significant only when it is set and the Monitor Mode Interrupt option is present.

If Bit 39 of n+2 is set and this bit is clear, monitor mode 1 is selected and only the memory parity error interrupt flag can be set while in monitor mode.

If Bit 39 of n+2 and this bit are both set, monitor mode 2 is in effect and the PC interrupt, MCU interrupt, I/O interrupt, and normal exit flags cannot be set.

n	E	S	R	B		P			A0
n+1	RA					BA			A1
n+2						LA		M	A2
n+3					XA	VL	F		A3
n+4									A4
n+5									A5
n+6									A6
n+7									A7
n+8	SO								
n+9	S1								
n+10	S2								
n+11	S3								
n+12	S4								
n+13	S5								
n+14	S6								
n+15	S7								
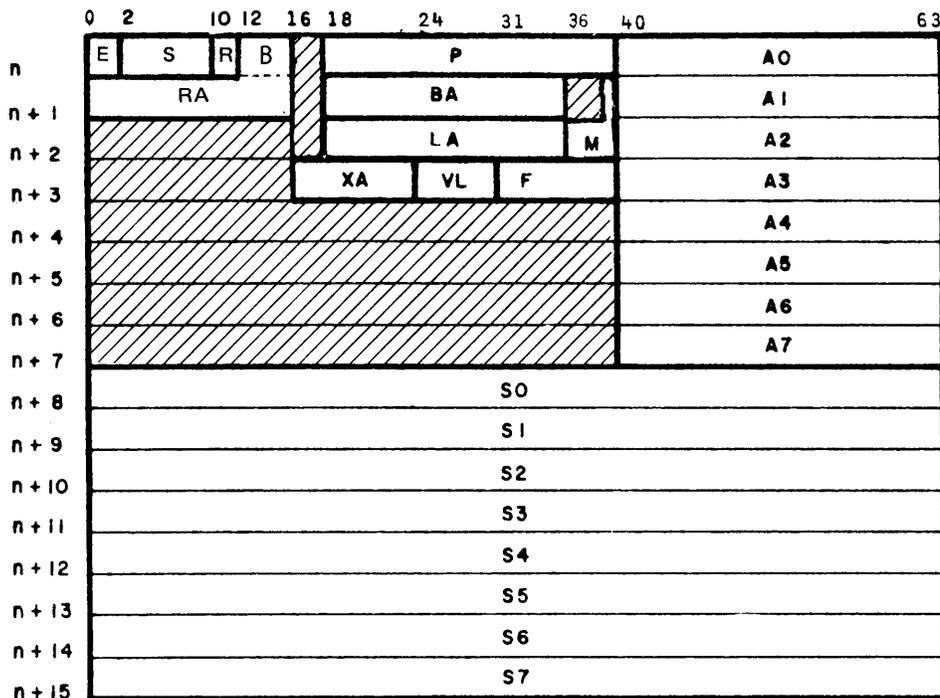
Registers

S	Syndrome bits
RAB	Read address for error (where B is bank)
P	Program address
BA	Base address
LA	Limit address
XA	Exchange address
VL	Vector length

E - Error type (bits 0,1 of n)

10 Uncorrectable memory

01 Correctable memory

R - Read mode (bits 10,11 of n)

00 Scalar

01 I/O

10 Vector

11 Fetch

M - Modes

n+1	39	Interrupt monitor mode[†]
n+2	36	Interrupt on correctable memory error
n+2	37	Interrupt on floating point error
n+2	38	Interrupt on uncorrectable memory error
n+2	39	Monitor mode

F - Flags

n+3	31	Programmable clock interrupt[††]
n+3	32	MCU interrupt
n+3	33	Floating point error
n+3	34	Operand range error
n+3	35	Program range error
n+3	36	Memory error
n+3	37	I/O interrupt
n+3	38	Error exit
n+3	39	Normal exit

† Supports Monitor Mode Interrupt option.

†† Supports Programmable Clock option.

Figure 3-8. Exchange package

n+2	Bit 36	Correctable memory error mode flag. When this bit is set, interrupts on correctable errors are enabled.
n+2	Bit 37	Floating point error mode flag. When this bit is set, interrupts on floating point errors are enabled.
n+2	Bit 38	Uncorrectable memory error mode flag. When this bit is set, interrupts on uncorrectable memory errors are enabled.
n+2	Bit 39	Monitor mode flag. When this bit is set and the Monitor Mode Interrupt Option is not present, all interrupts other than memory errors are inhibited. When the Monitor Mode Interrupt Option is present, this bit serves as the monitor mode select flag. When it is set, monitor mode 1 or monitor mode 2 is selected depending on the state of the interrupt monitor mode select bit (Bit 39 of n+1). The interrupt monitor mode select bit determines which interrupt flags can be set while the CPU is in monitor mode.

Bit 37 of n+2, the floating point error mode select, can be set or cleared during the execution interval for a program through use of the 0021 and 0022 instructions, respectively. Bits 38 and 39 of n+2 are not altered during the execution interval for the exchange package. Either of these bits can be altered only when the exchange package is inactive in memory.

F REGISTER

The F (flag) register is a nine-bit register that contains part of the exchange package for the currently active program. This register contains nine flags which are individually identified with the exchange package in figure 3-8. Setting any of these flags causes interruption of the program execution. When one or more flags are set, a request interrupt signal is sent to initiate an exchange sequence. The content of the F register is stored along with the rest of the exchange package and the monitor program can analyze the nine flags for the cause of the interruption. Before the monitor program exchanges back to the package, it may clear the flags in the F register area of the package. If any of the flag bits is set during the transfer of the exchange package to the CPU, another exchange will occur immediately.

Monitor mode interrupt option not present

Any flag other than the memory error flag, can be set in the F register
only if the currently active exchange package is *not* in monitor mode.
This means that these flags will set only if the highest order bit of
the M register is zero. With the exception of the memory error flag, if
the program is in monitor mode and the conditions for setting an F register
flag are otherwise present, the flag remains cleared and no exchange
sequence is initiated.

Monitor mode interrupt option present

If the monitor mode interrupt option is present and the currently active
exchange package is not in monitor mode (Bit 39 of n+2 of the M register
is zero), any of the nine F register flags can be set provided that all
interrupts are enabled.

If the program is in monitor mode 1 (Bit 39 of n+2 of the M register is
set and Bit 39 of n+1 of the M register is zero), the memory error flag is
the only one of the nine F register flags that can be set. The memory
error flag can be set while in monitor mode 1 if either of the two memory
parity error mode bits (Bits 36 and 38 of the M register) is also set.
When in monitor mode 1, none of the F register flags can be set but an
exchange sequence can be initiated by a 000 or a 004 instruction even
though the associated error exit flag or normal exit flag is not set.

If the program is in monitor mode 2 (Bits 39 of both n+1 and n+2 of the M
register are both set), all F register flags other than the PC interrupt,
MCU interrupt, I/O interrupt, and normal exit flags can be set and an
exchange sequence will be initiated.

EXCHANGE PACKAGE

An exchange package is a 16-word block of data in memory which is associated
with a particular computer program. It contains the basic parameters
necessary to provide continuity from one execution interval for the program
to the next. These parameters consist of the following:

    Program address register (P) - 22 bits
    Base address register (BA) - 18 bits
    Limit address register (LA) - 18 bits

Mode register (M) - 4 bits without MMI option; 5 bits with option
Exchange address register (XA) - 8 bits
Vector length register (VL) - 7 bits
Flag register (F) - 9 bits
Current contents of the eight A registers
Current contents of the eight S registers

The exchange package contents are arranged in a 16-word block as shown in figure 3-8. Data is swapped from memory to the computer operating registers and back to memory by the exchange sequence. This sequence exchanges the data in a currently active exchange package, which is residing in the operating registers, with an inactive exchange package in memory. The XA address of the currently active exchange package specifies the address of the inactive exchange package to be used in the swap. The data is exchanged and a new program execution interval is initiated by the exchange sequence.

The B register, T register, and V register contents are not swapped in the exchange sequence. The data in these registers must be stored and replaced as required by specific coding in the monitor program which supervises the object program execution.

Memory error data

Two bits in the Mode (M) register determine whether or not the exchange package contains data relevant to a memory error if one occurs prior to an exchange sequence. These are bit 36, the "Interrupt on correctable memory error bit" and bit 38, the "Interrupt on uncorrectable memory error bit". The error data, consisting of four fields of information, appears in the exchange package if bit 38 is set and an uncorrectable memory error is detected or if bit 36 is set and correctable memory error is encountered.

Error type (E) - The type of error encountered, uncorrectable or correctable, is indicated in bits 0 and 1 of the first word of the exchange package. Bit 0 is set for an uncorrectable memory error; bit 1 is set for a correctable memory error.

<u>Syndrome (S)</u> - The eight syndrome bits used in detecting the error are returned in bits 2 through 9 of the first word of the exchange package. Refer to section 5 for additional information.

<u>Read mode (R)</u> - This field indicates the read mode in progress when the error occurred and consists of bits 10 and 11 of the first word of the exchange package. These bits assume the following values:

      00   Scalar
      01   I/O
      10   Vector
      11   Instruction fetch

<u>Read address (RAB)</u> - The RAB field contains the address at which the error occurred. Bits 12 through 15 (B) of the first word of the exchange package contain bits 2^3 through 2^0 of the address and may be considered as the bank address; bits 0 through 15 (RA) of the second word of the exchange package contain bits 2^{19} through 2^4 of the address.

<u>Active exchange package</u>
An active exchange package is an exchange package which is currently residing in the computer operating registers. The interval of time in which the exchange package is active is called the execution interval for the exchange package and also for the program with which it is associated. The execution interval begins with an exchange sequence in which the subject exchange package moves from memory to the operating registers. The execution interval ends as the exchange package moves back to memory in a subsequent exchange sequence.

EXCHANGE SEQUENCE

The exchange sequence is the vehicle for moving an inactive exchange package from memory into the operating registers and at the same time moving the currently active exchange package from the operating registers back into memory. This swapping operation is done in a fixed sequence when all computational activity associated with the currently active

exchange package has stopped. The same 16-word block of memory is used as the source of the inactive exchange package and the destination of the currently active exchange package. The location of this block is specified by the content of the exchange address register and is a part of the currently active exchange package. The exchange sequence may be initiated in three different ways.

1. Dead start sequence
2. Interrupt flag set
3. Program exit

Initiated by dead start sequence

The dead start sequence forces the exchange address register content to zero and also forces a 000 code in the NIP register. These two actions cause the execution of a program error exit using memory address zero as the location of the exchange package. The inactive exchange package at address zero is then moved into the operating registers and a program is initiated using these parameters. The exchange package stored at address zero is largely noise as a result of the dead start operation and should be discarded by the subsequent entry of new data at these storage addresses.

Initiated by interrupt flag set

An exchange sequence can be initiated by setting any one of the nine interrupt flags in the F register. One or more flags set result in a request interrupt signal which initiates an exchange sequence.

Initiated by program exit

There are two program exit instructions that cause the initiation of an exchange sequence. The timing of the instruction execution (50 CPs) is the same in either case and consists of an exchange sequence and a fetch operation. They differ only in which of the two flags in the F register is set. The two instructions are:

Program code 000 - Error exit
Program code 004 - Normal exit

The two exits provide a means for a program to request its own termination. A non-monitor (object) program will usually use the normal exit instruction. to exchange back to the monitor program. The error exit allows for termination of an object program that has branched into an unused area of memory or into a data area. The exchange address selected is the same as for a normal exit.

There is a flag in the F register for each of these instructions. The appropriate flag is set providing the currently active exchange package is not in monitor mode. The inactive exchange package called in this case is normally one that executes in monitor mode and the flags are read from memory for evaluation of the cause of program termination.

The monitor program selects an inactive exchange package for activation by setting the address of the inactive exchange package into the XA register and then executing a normal exit instruction.

Exchange sequence issue conditions

An exchange sequence initiated by other than a 000 or 004 instruction has the following hold issue conditions, execution time, and special cases. The corresponding information for the 000 and 004 instructions is provided with the instruction descriptions in Section 4 of this manual.
Hold issue conditions:
        Instruction buffer data invalid
        NIP not blank
        Wait exchange flag not set
        S, V, or A registers busy

Execution time: 49 CPs; consists of an exchange sequence and a fetch
                  operation.

Special cases:
        Block instruction issue
        Block I/O references
        Block fetch

EXCHANGE PACKAGE MANAGEMENT

Each 16-word exchange package resides in an area defined during system
dead start that must lie within the lower 4096 words of memory. The
package at address 0 is that of the monitor program. Other packages
provide for object programs and monitor tasks. These packages lie
outside of the field lengths for the programs they represent as
determined by the base and limit addresses for the programs. Only the
monitor program has a field defined so that it can access all of memory
including the exchange package areas. This allows the monitor program
to define or alter all exchange packages other than its own when it is
the currently active exchange package.

Proper management of exchange packages dictates that a non-monitor
program always exchange back to the monitor program that exchanged to
it. This assures that the program information is always swapped back
into its proper exchange package.

Consider the case where exchange packages exist for programs A, B, and C.
Program A is the monitor program, program B is a user program, and program
C is an interrupt processing program.

The monitor program, A, begins an execution interval following dead start.
No interrupts can terminate its execution interval since it is in monitor
mode[†]. The monitor program voluntarily exits by issuing a 004 exit
instruction. Before doing so, however, it sets the contents of the XA
register to point to B's exchange package so that B will be the next
program to execute and it sets the exit address in B's exchange package
to point back to the monitor.

The exchange sequence to B causes the exit address from B's exchange
package to be entered in the XA register. At the same time, the exchange
address in the XA register goes to B's exchange package area along with all
other program parameters for the monitor program. When the exchange is
complete, program B begins its execution interval.

---

† Assumes Monitor Mode Interrupt Option is not present. Refer to descrip-
  tion of M register.

Suppose further that while B is executing, an interrupt flag sets
initiating an exchange sequence. Since B cannot alter the XA register,
the exit is back to the monitor program. Program B's parameters swap back
into B's exchange package area; the monitor program parameters held in
B's package during the execution interval swap back into the operating
registers.

The monitor, upon resuming execution, determines that an interrupt has
caused the exchange and sets the XA register to call the proper interrupt
processor into execution. It does this by setting XA to point to the
exchange package for program C. Then, it clears the interrupt and
initiates execution of C by executing a 004 exit instruction. Depending
on the design of the operating system, the interrupt processor program
could execute in monitor mode or in user mode.

MEMORY FIELD PROTECTION

Each object program at execution time has a designated field of memory
holding instructions and data. The field limits are specified by the
monitor program when the object program is loaded and initiated. The
field may begin at any word address that is a multiple of 16 and may
continue to another address that is also a multiple of 16. The field
limits are contained in two registers, the base address register (BA)
and the limit address register (LA), which are described later in this
subsection.

All memory addresses contained in the object program code are relative
to the base address which begins the defined field. It is, therefore,
not possible for an object program to read or alter any memory location
with a lower absolute address than the base address. Each object program
reference to memory is also checked against the limit address to determine
if the address is within the bounds assigned. A memory reference beyond
the assigned field limit is prevented from reading or altering the memory
content and for a non-monitor mode program, creates an error condition that
terminates program execution. The program or operand range flag is set

to indicate the error correction. The monitor program upon resuming
execution determines the cause of the interrupt and takes appropriate
action, perhaps terminating the user program.

BA REGISTER

The 18-bit BA register holds the base address of the user field during
the execution interval for each exchange package. The contents of this
register are interpreted as the upper 18 bits of a 22-bit memory address.
The lower four bits of the address are assumed zero. Absolute memory
addresses are formed by adding (BA) * 16 to the relative address specified
by the CPU instructions. The BA register always indicates a bank 0
memory address.

LA REGISTER

The 18-bit LA register holds the limit address of the user field during
the execution interval for each exchange package. The contents of LA
are interpreted as the upper 18 bits of a 22-bit memory address. The
lower four bits of the address are assumed zero. The LA register always
indicates a bank 0 memory address.

The final address that can be executed or referenced by a program is at
$[(LA) \times 2^4] - 1$. Note that the (LA) is absolute, not relative; it is not
added to (BA).

DEAD START SEQUENCE

The dead start sequence is that sequence of operations required to start
a program running in the CPU after power has been turned off and then
turned on again. All registers in the machine, all control latches,
and all words in memory are assumed to be invalid after power has been
turned on. The sequence of operations required to begin a program is
initiated by the maintenance control unit. This unit sequences the
following operations:
   1. Turns on master clear signal.
   2. Turns on I/O clear signal.

3. Turns off I/O clear signal.
4. Loads memory via MCU channel.
5. Turns off master clear signal.

The master clear signal stops all internal computation and forces the critical control latches to predetermined states. The I/O clear signal clears the input channel address register of the channel connected to the MCU and activates the input channel conected to the MCU subsystem. All other input channels remain inactive. The maintenance control unit then loads an initial exchange package and monitor program. The exchange package must be located at address zero in memory. Turning off the master clear signal initiates the exchange sequence to read this package and to begin execution of the monitor program. Subsequent actions are dictated by the design of the operating system.

SECTION 4

INSTRUCTIONS

INSTRUCTION FORMAT

Each instruction is either a one-parcel (16-bit) instruction or a two-parcel (32-bit) instruction. Instructions are packed four parcels per word. Parcels in a word are numbered from left to right as 0 through 3 and can be addressed in branch instructions. A two-parcel instruction may begin in any parcel of a word and may span a word boundary. A two-parcel instruction that begins in the fourth parcel of a word ends in the first parcel of the next word. No padding to word boundaries is required.

Instructions have the following general form:

```
| 4 , 3, 3, 3, 3|        16        |
  g   h  i  j  k           m
|←First parcel→|←Second parcel→|
```
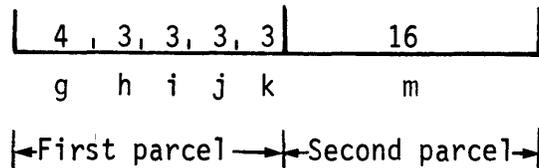
Figure 4-1. General format for instructions

Five variants of this general format use the fields in different ways. Two of these variant forms are two-parcel formats, two are one-parcel formats, and one is either a one-parcel or a two-parcel format.

ARITHMETIC, LOGICAL FORMAT

For arithmetic and logical instructions, a 7-bit operation code (gh) is followed by three 3-bit address fields. The first field, i, designates the result register. The j and k fields designate the two operand registers or are combined to designate a 6-bit B or T register address. This format is illustrated in figure 4-2.

g   h   i   j   k

| 4 | 3 | 3 | 3 | 3 |

OPERATION
CODE

RESULT
REG.

OPERAND
REG.

OPERAND
REG.

I6 BITS
ARITHMETIC, LOGICAL
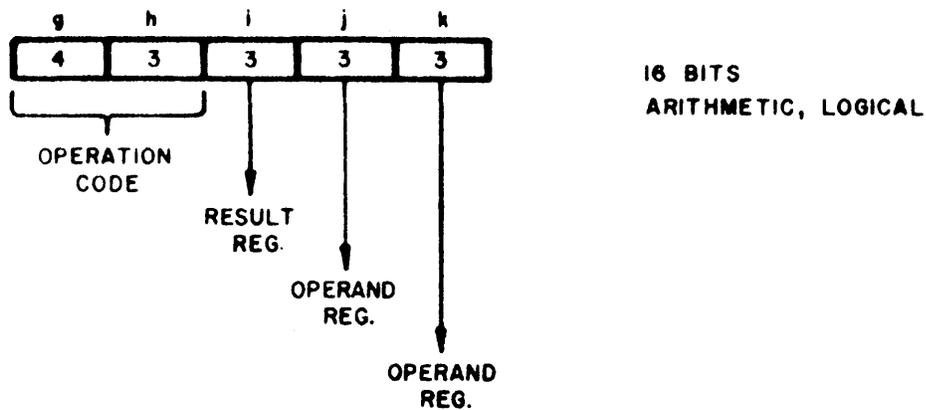
Figure 4-2. Format for arithmetic and
logical instructions

SHIFT, MASK FORMAT

The shift and mask instructions consist of a 7-bit operation code (gh)
followed by a 3-bit field and a 6-bit field. The 3-bit i field desig-
nates the result and operand registers. The 6-bit combined jk field
specifies a shift or mask count. This format is illustrated in figure 4-3.

g   h   i   j k

| 4 | 3 | 3 | 6 |

OPERATION
CODE

OPERAND AND
RESULT REG.

SHIFT, MASK COUNT

I6 BITS
SHIFT, MASK
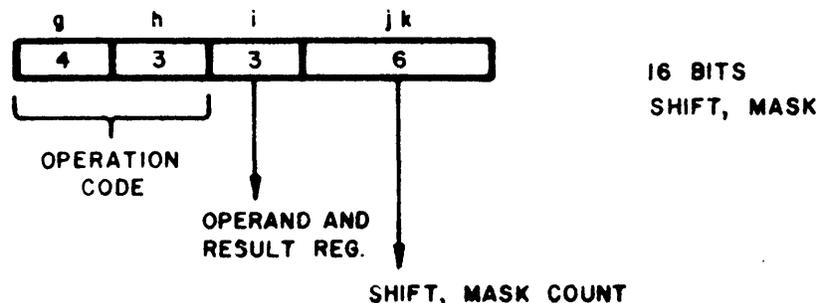
Figure 4-3. Format for shift and mask
instructions

IMMEDIATE CONSTANT FORMAT

The instructions that enter immediate constants into A registers have
either a one-parcel or a two-parcel form. Only the two-parcel form exists
for entering immediate constants into S registers. For the one-parcel
form, the j and k fields are combined to give a 6-bit quantity. For the

two-parcel form, the j, k, and m fields are combined to give a 22-bit quantity. In either form, a 7-bit operation code (gh) and a 3-bit result field designating a result register precede the immediate constant. Figure 4-4 illustrates the instruction format for immediate constant instructions.
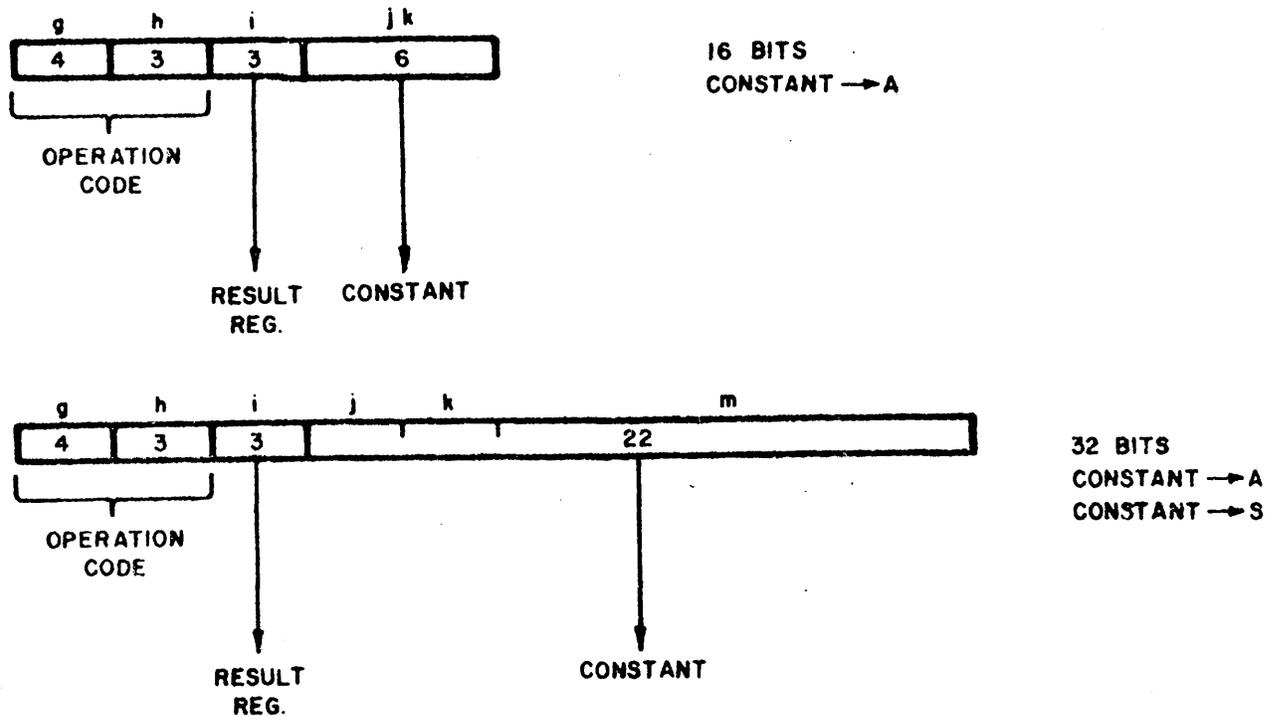
Figure 4-4. Format for immediate constant instructions

MEMORY TRANSFER FORMAT

Instructions that transfer data between the A or S registers and memory require a 32-bit format. For these instructions, a 4-bit operation code (g) is followed by two 3-bit fields and a 22-bit field. The first 3-bit field (h) designates an index (A) register.

When the h field is zero, the special value of zero is considered to be the address index. Contents of Ah are not affected. The second 3-bit field (i) designates a result or source register. The 22-bit field formed by j, k, and m, specifies a memory word address. The upper two bits of the j field are unused. An operand range error occurs if either bit is set.

Figure 4-5 illustrates the format of memory transfer instructions.
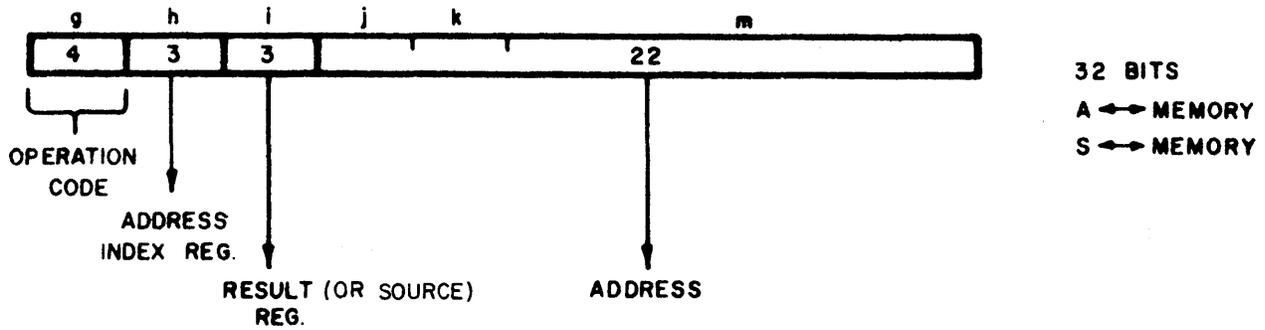
Figure 4-5. Format for memory transfer instructions

BRANCH FORMAT

In general, the branch instructions are two-parcel instructions. A 7-bit operation code (gh) is followed by a 25-bit field formed by combining i, j, k, and m. The 25-bit field contains a parcel address and allows branching to a quarter-word boundary. The 3-bit i field is unused. A program range error occurs if either of the two low-order bits of i is set; the high-order bit of i is ignored.

Figure 4-6 illustrates the two-parcel format for branch instructions.
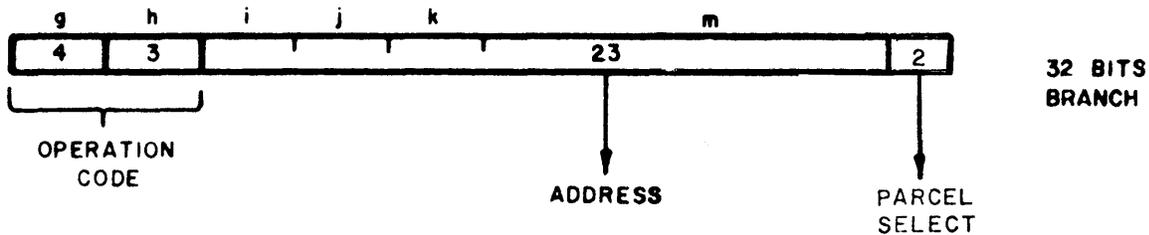
Figure 4-6. Two-parcel format for branch instructions

The unconditional branch to (Bjk) instruction requires only one parcel. For this instruction, there is a 7-bit operation code (gh) followed by a null i field and a combined jk field which specifies a B register that contains a parcel address. The format is not illustrated.

SPECIAL REGISTER VALUES

The S_0 and A_0 registers provide special values when referenced in the j or k fields of an instruction. In these cases, the special value is used as the operand and the actual value of the S_0 or A_0 register is ignored. Such a use does not alter the actual value of the S_0 or A_0 register. If S_0 or A_0 is used in the i field, the actual value of the register is provided as the operand.

Field	Operand value
Ai, i = 0	(A_0)
Aj, j = 0	0
Ak, k = 0	1
Si, i = 0	(S_0)
Sj, j = 0	0
Sk, k = 0	2^{63}
Ah, h = 0	0

INSTRUCTION ISSUE

Instructions are read a parcel at a time from the instruction buffers and delivered to the NIP register. The instruction issues and is passed to the CIP register when the conditions in the functional unit and registers are such that the functions required for execution may be performed without conflicting with a previously issued instruction. Instruction parcels may issue at a maximum rate of one per clock period. Once an instruction has been delivered to the CIP it is considered as issued and it must be completed in a fixed time frame following its final clock period in the CIP register. No delays are allowed from issue to delivery of data to the destination operating registers.

Entry to the NIP is blocked for the second half of a two-parcel instruction. The parcel is delivered to the LIP register, instead. The blank NIP for the second parcel is issued as a do-nothing instruction in the CIP.

INSTRUCTION DESCRIPTIONS

This section contains detailed information about individual instructions
or groups of related instructions. Descriptions are presented in the
octal code sequence defined by the gh fields. Each subsection begins
with boxed information consisting of the format and a brief summary of
each instruction described in the subsection. The appearance of an m
in a format designates that the instruction consists of two parcels.
An x in the format signifies that the field containing the x is ignored
during instruction execution.

Following the header information is a more detailed description of the
instruction or instructions, including a list of hold issue
conditions, execution time, and special cases. Hold issue conditions
refer to those conditions that delay issue of an instruction until the
conditions are met.

Instruction issue time assumes that if an instruction issues at clock
period n, the next instruction will issue at clock period n + issue time
if its issue conditions have been met.

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                   │
│     000xxx      Error exit                                        │
│                                                                   │
└─────────────────────────────────────────────────────────────────┘
```

This instruction is treated as an error condition and an exchange
sequence occurs. The content of the instruction buffers is voided
by the exchange sequence. If monitor mode is not in effect, the
error exit flag in the F register is set. All instructions issued
prior to this instruction are run to completion. When the results
of previously issued instructions have arrived at the operating
registers, an exchange occurs to the exchange package designated by
the contents of the XA register. The program address stored in the
exchange package on the terminating exchange sequence is advanced by
one count from the address of the error exit instruction. The error
exit instruction is not generally used in program code. Its purpose
is to halt execution of an incorrectly coded program that branches
into an unused area of memory or into a data area.

Hold issue conditions
        034 - 037 in process
        Exchange in process

Execution time
        Instruction issue 50 CPs; this time includes an exchange sequence
        (36 CPs) and a fetch operation (14 CPs).

Special cases
        None

```
--------------------------------------------------------------------------
:                                                                          :
:   001ijk     Monitor functions                                          :
:                                                                          :
--------------------------------------------------------------------------
```

This instruction is privileged to monitor mode and performs specialized
functions useful to the operating system. Functions are selected through
the i designator. The instruction is treated as a pass instruction if the
monitor mode bit is not set or if the i designator is 5, 6, or 7.

Subfunctions defined by the i designator are as follows:

        0010jk   Set the current address (CA) register for the channel
                  indicated by (Aj) to (Ak) and **activate the channel**
        0011jk   Set the limit address (CL) register for the channel
                  indicated by (Aj) to (Ak)
        0012jx   Clear the interrupt flag and error flag for the
                  channel indicated by (Aj) and/or deactivate the channel
        0013jx   Enter the XA register with (Aj)
        0014jx   Enter the real-time clock register with (Sj)

When the i designator is 0, 1, or 2, the instruction controls the
operation of the I/O channels. Each channel has two registers that
direct the channel activity. The CA register for a channel contains
the address of the current channel word. The CL register specifies
the limit address. In programming the channel, the CL register is
initialized and setting CA activates the channel. As the transfer
continues, CA is incremented toward CL. When (CA) = (CL), the
transfer is complete for words at initial (CA) through (CL)-1.
When the j designator is 0 or when the content of Aj is less than 2
or greater than 25, the functions are executed as pass instructions.
When the k designator is 0, CA or CL is set to 1.

When the i designator is 3, the instruction transmits bits 2^{11} through
2^4 of (Aj) to the exchange address (XA) register. When the j designator
is 0, the XA register is cleared.

When the i designator is 4, the instruction transmits the contents of Sj to the real-time clock register. When the j designator is 0, the real-time clock is cleared.

If the Programmable Clock Interrupt (PCI) Option is installed, the content of the k field is relevant for this instruction.

Hold issue conditions

034 - 037 in process

Exchange in process

For 0010, 0011, 0012, 0013, and 0014, Aj or Sj or Ak Reserved

Execution time

Instruction issue 1 CP

Special cases

If the program is not in monitor mode, instruction becomes a no-op although all hold issue conditions remain effective.

For 0010, 0011, and 0012:

If j = 0, instruction is a no-op

If $(Aj) < 2$ or $(Aj) \geq 31_8$, instruction is a no-op

If k = 0, CA or CL is set to 1

For 0013:

If j = 0, XA register is cleared

For 0014:

If j = 0, RTC register is cleared

Correct priority interrupting channel number can be read (via 033 instruction) 2 CP after issue of 0012 instruction.

```
--------------------------------------------------------------------------------
|                                                                              |
|     0014jk  Programmable clock interrupt functions                           |
|                                                                              |
--------------------------------------------------------------------------------
```

When the Programmable Clock Interrupt Option is installed, subfunctions
of the 0014 monitor mode instruction defined by the k designator are
recognized. When the Programmable Clock Option is not installed, none of
these subfunctions is recognized and the instruction is always interpreted
as an enter real-time clock register instruction.

The following subfunctions are defined by the k designator:

    0014j0   Enter the real-time clock register with (Sj)
    0014j4   Enter interrupt interval (II) register with (Sj)
    0014j5   Clear the programmable clock interrupt request
    0014j6   Enable programmable clock interrupt request
    0014j7   Disable programmable clock interrupt requests

When the k designator is 0, this instruction loads the contents of the Sj
register into the real-time clock (RTC) register. When the j designator
is 0, the real-time clock register is cleared.

When the k designator is 4, this instruction loads the lower 32 bits
from the Sj register into both the Interrupt Interval (II) register and
the Interrupt Countdown (ICD) counter.

When the k designator is 5, this instruction clears the programmable clock
interrupt request if the request was previously set by an interrupt count
down to zero.

When the k designator is 6, this instruction enables repeated programmable
clock interrupt requests at a repetition rate determined by the value
stored in the Interrupt Interval (II) register.

When the k designator is 7, this instruction disables repeated programmable
clock interrupt requests until a 0014j6 instruction is executed to enable
the requests.

Refer to section 6 for additional information about the Programmable Clock
Interrupt Option.

Hold issue conditions

034 - 037 in process

Exchange in process

For 0014, Aj or Sj or Ak reserved

Execution time

Instruction issue   1 CP

Special case

For 0014jk:

If the program is not in monitor mode, instruction becomes a no-op but all hold issue conditions remain effective.

```
----------------------------------------------------------------------
|                                                                    |
|    0020xk    Transmit (Ak) to VL                                   |
|                                                                    |
----------------------------------------------------------------------
```

This instruction enters the vector length (VL) register with a value
determined by the contents of Ak. The low order seven bits of (Ak)
are entered into the VL register. The number of operations performed is
determined by first subtracting one from the contents of VL and then
adding one to the low-order six bits of the result. For example, if
(VL) = 100_8, then 100-1 = 77 and 77+1 = 100. However, if (VL) = 0,
then 0-1 = 177 and 77+1 = 100. Thus, the number of vector operations is
64 when the content of Ak is 0 or 64 before executing the 0020 instruction.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        Ak reserved

Execution time
        Instruction issue   1 CP
        VL register ready   1 CP

Special cases
        Maximum vector length is 64
        (Ak) = 1 if k = 0
        (VL) = 0 if k $\neq$ 0 and (Ak) = 0

```
+-------------------------------------------------------------------+
|                                                                   |
|    0021xx      Set floating point mode flag in M register         |
|    0022xx      Clear floating point mode flag in M register       |
|                                                                   |
+-------------------------------------------------------------------+
```

These instructions set (0021xx) or clear (0022xx) the floating point mode flag in the M register. They do not check the previous state of the flag (there is no way of testing the flag).

When set, the floating point mode flag enables interrupts on floating point overflow errors as described in Section 3.

Hold issue conditions

     034 - 037 in process

     Exchange in process

     Ak reserved

Execution time

     Instruction issue 1 CP

Special cases

     None

```
:---------------------------------------------------------------------:
:                                                                     :
:    003xjx    Transmit (Sj) to vector mask                           :
:                                                                     :
:---------------------------------------------------------------------:
```

This instruction enters the vector mask (VM) register with the contents
of Sj. The VM register is cleared if the j designator is zero. This
instruction is used in conjunction with the vector merge instructions
(146 and 147) in which an operation is performed depending on the
contents of VM.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        Sj reserved
        003 in process - unit busy 3 CPs
        14x in process - unit busy (VL) + 4 CPs
        175 in process - unit busy (VL) + 4 CPs

Execution time
        Instruction issue 1 CP
        VM ready in 3 CPs except for use in 073 instruction
        For 073 instruction, VM ready in 6 CPs

Special cases
        (Sj) = 0 if j = 0

```
┌─────────────────────────────────────────────────────────────────────────┐
│                                                                           │
│     004xxx     Normal exit                                                │
│                                                                           │
└─────────────────────────────────────────────────────────────────────────┘
```

This instruction causes an exchange sequence. The contents of the
instruction buffers are voided by the exchange sequence. If monitor
mode is not in effect, the normal exit flag in the F register is set.
All instructions issued prior to this instruction are run to completion.
When all results have arrived at the operating registers as a result
of previously issued instructions, an exchange sequence occurs to the
exchange package designated by the contents of the XA register. The
program address stored in the exchange package is advanced one count
from the address of the normal exit instruction. This instruction is
used to issue a monitor request from a user program.

Hold issue conditions
        034 - 037 in process
        Exchange in process

Execution time
        Instruction issue 50 CPs; this time includes an exchange sequence
        (36 CPs) and a fetch operation (14 CPs).

Special cases
        Block instruction issue
        Begin exchange sequence

```
+-----------------------------------------------------------------+
|                                                                 |
|     005xjk      Branch to (Bjk)                                 |
|                                                                 |
+-----------------------------------------------------------------+
```

This instruction sets the P register to the parcel address specified
by the contents of Bjk causing execution to continue at that address.
The instruction is used to return from a subroutine.

Hold issue conditions

        034 - 037 in process

        Exchange in process

Execution time

        Instruction issue:

            Both parcels of branch in a buffer and branch address in a
            buffer   7 CPs

            Both parcels of branch in a buffer and branch address not
            in a buffer   16 CPs

            Second parcel of branch not in a buffer and branch address
            in a buffer   16 CPs

            Second parcel of branch not in a buffer and branch address
            not in a buffer   25 CPs

Special cases

        The parcel following an 005 instruction is not used for branching;

        however, it can cause a delay of the 005 instruction if it is

        out of buffer. See execution times.

```
+-------------------------------------------------------------------------+
|                                                                         |
|     006ijkm      Branch to ijkm                                         |
|                                                                         |
+-------------------------------------------------------------------------+
```

This two-parcel instruction sets the P register to the parcel address
specified by the low order 22 bits of the ijkm field. Execution
continues at that address. The high order bit of the ijkm field is
ignored.

Hold issue conditions

      034 - 037 in process

      Exchange in process

Execution time

      Instruction issue:

            Both parcels of branch in the same buffer and branch address
            in a buffer   5 CPs

            Both parcels of branch in the same buffer and branch address
            not in a buffer   14 CPs

            Both parcels of branch in different buffers and branch
            address in a buffer   7 CPs

            Both parcels of branch in different buffers and branch
            address not in a buffer   16 CPs

            Second parcel of branch not in a buffer and branch address
            in a buffer   16 CPs

            Second parcel of branch not in a buffer and branch address
            not in a buffer   25 CPs

Special cases

      None

```
-----------------------------------------------------------------------
|                                                                     |
|      007ijkm    Return jump to ijkm; set $B_{00}$ to (P)            |
|                                                                     |
-----------------------------------------------------------------------
```

This two-parcel instruction sets register B_{00} to the address of the
following parcel. The P register is then set to the parcel address
specified by the low order 22 bits of the ijkm field. Execution
continues at that address. The high order bit of the ijkm field is
ignored. The purpose of this instruction is to provide a return
linkage for subroutine calls. The subroutine is entered via a
return jump. The subroutine returns to the caller at the instruction
following the call by executing a branch to the contents of a
B register.

Hold issue conditions

    034 - 037 in process

    Exchange in process

Execution time

    Instruction issue:

        Both parcels of branch in the same buffer and branch address
        in a buffer   5 CPs

        Both parcels of branch in the same buffer and branch address
        not in a buffer   14 CPs

        Both parcels of branch in different buffers and branch
        address in a buffer   7 CPs

        Both parcels of branch in different buffers and branch
        address not in a buffer   16 CPs

        Second parcel of branch not in a buffer and branch address
        in a buffer   16 CPs

        Second parcel of branch not in a buffer and branch address
        not in a buffer   25 CPs

Special cases

    None

```
    ┌─────────────────────────────────────────────────────────────────────┐
    │  010ijkm    Branch to ijkm if (A₀) = 0                                │
    │  011ijkm    Branch to ijkm if (A₀) ≠ 0                                │
    │  012ijkm    Branch to ijkm if (A₀) positive                          │
    │  013ijkm    Branch to ijkm if (A₀) negative                          │
    └─────────────────────────────────────────────────────────────────────┘
```

These two-parcel instructions test the contents of A_0 for the
condition specified by the h field. If the condition is satisfied,
the P register is set to the parcel address specified by the low order
22 bits of the ijkm field and execution continues at that address.
The high order bit of the ijkm field is ignored. If the condition is
not satisfied, execution continues with the instruction following the
branch instruction.

Hold issue conditions

    034 - 037 in process

    Exchange in process

    A_0 busy in last 2 CPs

Execution time

    Instruction issue:

        Both parcels of branch in the same buffer and branch address
        in a buffer   5 CPs

        Both parcels of branch in the same buffer and branch address
        not in a buffer   14 CPs

        Both parcels of branch in different buffers and branch
        address in a buffer   7 CPs

        Both parcels of branch in different buffers and branch
        address not in a buffer   16 CPs

        Second parcel of branch not in a buffer and branch address
        in a buffer   16 CPs

        Second parcel of branch not in a buffer and branch address
        not in a buffer   25 CPs

        Both parcels of branch in the same buffer and branch not taken   2 CPs

        Both parcels of branch in different buffers and branch not taken   4 CPs

        Second parcel of branch not in a buffer and branch not taken   13 CPs

Special cases

    (A_0) = 0 is considered a positive condition

2240004   4-19   E

```
014ijkm    Branch to ijkm if (S_0) = 0
015ijkm    Branch to ijkm if (S_0) ≠ 0
016ijkm    Branch to ijkm if (S_0) positive
017ijkm    Branch to ijkm if (S_0) negative
```

These two-parcel instructions test the contents of S_0 for the condition specified by the h field. If the condition is satisfied, the P register is set to the parcel address specified by the low order 22 bits of the ijkm field and execution continues at that address. The high order bit of the ijkm field is ignored. If the condition is not satisfied, execution continues with the instruction following the branch instruction.

Hold issue conditions

034 - 037 in process

Exchange in process

S_0 busy in last 2 CPs

Execution time

Instruction issue:

Both parcels of branch in the same buffer and branch address in a buffer   5 CPs

Both parcels of branch in the same buffer and branch address not in a buffer   14 CPs

Both parcels of branch in different buffers and branch address in a buffer   7 CPs

Both parcels of branch in different buffers and branch address not in a buffer   16 CPs

Second parcel of branch not in a buffer and branch address in a buffer   16 CPs

Second parcel of branch not in a buffer and branch address not in a buffer   25 CPs

Both parcels of branch in the same buffer and branch not taken   2 CPs

Both parcels of branch in different buffers and branch not taken   4 CPs

Second parcel of branch not in a buffer and branch not taken   13 CPs

Special cases

$(S_0) = 0$ is considered a positive condition

```
---------------------------------------------------------------------
|                                                                   |
|    020ijkm    Transmit jkm to Ai                                  |
|    021ijkm    Transmit complement of jkm to Ai                    |
|                                                                   |
---------------------------------------------------------------------
```

The 020 instruction enters into Ai a 24-bit value that is composed of
the 22-bit jkm field and two upper bits of zero.

The 021 instruction enters into Ai a 24-bit value that is the complement
of a value formed by the 22-bit jkm field and two upper bits of zero. The
complement is formed by changing all one bits to zero and all zero bits to
one. Thus, for the 021 instruction, the upper two bits of Ai are set to one
and the instruction provides a means of entering a negative value into Ai.
The instructions are both two-parcel instructions.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        A register access conflict
        Ai reserved

Execution time
        Instruction issue:
            Both parcels in same buffer   2 CPs
            Parcels in different buffers   4 CPs
            Second parcel not in a buffer   13 CPs
        Ai ready   1 CP

Special cases
        None

```
+-------------------------------------------------------------------------+
:                                                                         :
:      022ijk      Transmit jk to Ai                                      :
:                                                                         :
+-------------------------------------------------------------------------+
```

This one-parcel instruction enters the 6-bit quantity from the jk field
into the low order 6 bits of Ai. The upper 18 bits of Ai are zeroed.
No sign extension occurs.

Hold isssue conditions
        034 - 037 in process
        Exchange in process
        A register access conflict
        Ai reserved

Execution time
        Instruction issue 1 CP
        Ai ready 1 CP

Special cases
        None

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                   │
│      023ijx    Transmit (Sj) to Ai                                │
│                                                                   │
└─────────────────────────────────────────────────────────────────┘
```

This instruction enters the low order 24 bits of (Sj) into Ai. The
high order bits of (Sj) are ignored.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        A register access conflict
        Ai reserved
        Sj reserved

Execution time
        Instruction issue 1 CP
        Ai ready 1 CP

Special cases
        (Sj) = 0 if j = 0

```
| 024ijk     Transmit (Bjk) to Ai
| 025ijk     Transmit (Ai) to Bjk
```

The 024 instruction enters the contents of Bjk into Ai.

The 025 instruction enters the contents of Ai into Bjk.

Hold issue conditions

      034 - 037 in process

      Exchange in process

      A register access conflict   (024 only)

      Ai reserved

Execution time

      For 024, Ai ready 1 CP

      Instruction issue for 024 or 025 1 CP

Special cases

      None

```
┌─────────────────────────────────────────────────────────────────────────┐
│   026ij0      Population count of (Sj) to Ai                               │
│   026ij1      Population count parity of (Sj) to Ai; requires presence     │
│               of Vector Population Instructions Option.                    │
└─────────────────────────────────────────────────────────────────────────┘
```

The 026ij0 instruction counts the number of bits set to one in (Sj) and
enters the result into the low order 7 bits of Ai. The upper 17 bits of
Ai are zeroed.

The 026ij1 instruction counts the number of bits set to one in (Sj).
Then, the least significant bit, which shows the odd/even state of the
result is transferred to the least significant bit position of the Ai
register. The actual population count is not transferred. This instruc-
tion is recognized only when the Vector Population Instructions Option is
installed; otherwise it operates as a 026ij0 instruction.

The instructions are executed in the population/leading zero count unit.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        A register access conflict
        Ai reserved
        Sj reserved

Execution time
        Instruction issue 1 CP
        Ai ready 4 CPs

Special cases
        (Ai) = 0 if j = 0

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                   │
│     027ijx    Leading zero count of (Sj) to Ai                    │
│                                                                   │
└─────────────────────────────────────────────────────────────────┘
```

This instruction counts the number of leading zeros in Sj and enters
the result into the low order seven bits of Ai. The upper 17 bits of
Ai are zeroed.
The instruction is executed in the population/leading zero count unit.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai reserved

Sj reserved

Execution time

Instruction issue 1 CP

Ai ready 3 CPs

Special cases

(Ai) = 64 if j = 0

(Ai) = 0 if (Sj) is negative

```
---------------------------------------------------------------
|                                                             |
|    030ijk     Integer sum of (Aj) and (Ak) to Ai           |
|    031ijk     Integer difference (Aj) and (Ak) to Ai       |
|                                                             |
---------------------------------------------------------------
```

These instructions are executed in the address add unit.

The 030 instruction forms the integer sum of (Aj) and (Ak) and enters the result into Ai. No overflow is detected.

The 031 instruction forms the integer difference of (Aj) and (Ak) and enters the result into Ai. No overflow is detected.

Hold issue conditions

     034 - 037 in process

     Exchange in process

     A register access conflict

     Ai, Aj, or Ak reserved

Execution time

     Instruction issue 1 CP

     Ai ready 2 CPs

Special cases

For 030:

     $(Ai) = (Ak)$ if $j = 0$ and $k \neq 0$

     $(Ai) = 1$ if $j = 0$ and $k = 0$

     $(Ai) = (Aj)+1$ if $j \neq 0$ and $k = 0$

For 031:

     $(Ai) = -(Ak)$ if $j = 0$ and $k \neq 0$

     $(Ai) = -1$ if $j = 0$ and $k = 0$

     $(Ai) = (Aj)-1$ if $j \neq 0$ and $k = 0$

```
---------------------------------------------------------------
|                                                             |
|    032ijk     Integer product of (Aj) and (Ak) to Ai        |
|                                                             |
---------------------------------------------------------------
```

This instruction forms the integer product of (Aj) and (Ak) and
enters the low order 24 bits of the result into Ai. No overflow
is detected.

This instruction is executed in the address multiply unit.

Hold issue conditions
     034 - 037 in process
     Exchange in process
     A register access conflict
     Ai, Aj, or Ak reserved

Execution time
     Instruction issue 1 CP
     Ai ready 6 CPs

Special cases
     (Ai) and (Aj) = 0 if j = 0
     (Ak) = 1 and (Ai) = (Aj) if k = 0 and j ≠ 0

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                   │
│     033ijk    Transmit I/O status to Ai                           │
│                                                                   │
└─────────────────────────────────────────────────────────────────┘
```

This instruction enters channel status information into Ai. The j
and k designators and the contents of Aj define the desired information.

        033i0x   Channel number of highest priority interrupt request
                  to Ai

        033ij0   Current address of channel (Aj) to Ai

        033ij1   Error flag of channel (Aj) to Ai

The channel number of the highest priority interrupt request is entered
into Ai when the j designator is zero. The contents of Aj specifies a
channel number when the j designator is nonzero. The value of the
current address (CA) register for the channel is entered into Ai when
the k designator is an even number. The error flag for the channel is
entered into the low order bit of Ai when the k designator is an odd
number. The high-order bits of Ai are cleared. The error flag can be
cleared only in monitor mode using the 0012 instruction.

The 033 instruction does not interfere with channel operation and is
not protected from user execution.

<u>Hold issue conditions</u>
      034 - 037 in process
      Exchange in process
      A register access conflict
      Ai reserved
      Aj reserved

<u>Execution time</u>
      Instruction issue 1 CP
      Ai ready 4 CPs

Special cases

(Ai) = highest priority channel causing interrupt if (Aj) = 0

(Ai) = current address of channel (Aj) if (Aj) ≠ 0 and k = 0,2,4,6

(Ai) = I/0 error flag of channel (Aj) if (Aj) ≠ 0 and k = 1,3,5,7

(Ai) = 0 if (Aj) = 1

2 CPs must elapse after an 0012xx instruction issue before issuing an 033i00 instruction.

```
+------------------------------------------------------------------------+
|   034ijk      Block transfer (Ai) words from memory starting at        |
|               address (A0) to B registers starting at register jk.     |
|   035ijk      Block transfer (Ai) words from B registers starting      |
|               at register jk to memory starting at address (A0)        |
|   036ijk      Block transfer (Ai) words from memory starting at        |
|               address (A0) to T registers starting at register jk      |
|   037ijk      Block transfer (Ai) words from T registers starting      |
|               at register jk to memory  starting at address (A0)       |
+------------------------------------------------------------------------+
```

These instructions perform block transfers between memory and B or T
registers.

In all of the instructions, the amount of data transferred is specified
by the lower seven bits of (Ai). See special cases for details.

The first register involved in the transfer is specified by jk. Successive
transfers involve successive B or T registers until B_{77} or T_{77} is reached.
Since processing of the registers is circular, B_{00} will be processed
after B_{77} and T_{00} will be processed after T_{77} if the count in (Ai) is
not exhausted.

The first memory location referenced by the transfer instruction is
specified by (A0). The A0 register contents are not altered by
execution of the instruction. Memory references are incremented by one
for successive transfers.

For transfers of B registers to memory, each 24-bit value is right adjusted
in the word; the upper 40 bits are zeroed. When transferring from memory
to B registers, only the low order 24 bits are transmitted; the upper 40
bits are ignored.

Hold issue conditions

    A_0 reserved

    Ai reserved

    Block sequence flag set (034 - 037, 176, 177)

    034 - 037 in process

    Exchange in process

    Scalar reference in CP2

    Rank B data valid

    Fetch request in last clock period

    I/O memory request

Execution time

For 034, 036:

    Instruction issue 14 CPs + (Ai) if (Ai) $\neq$ 0; 5 CPs if (Ai) = 0

For 035, 037:

    Instruction issue 6 CPs + (Ai) if (Ai) $\neq$ 0; 7 CPs if (Ai) = 0

Special cases

1. Block all issues when in process.

2. Block all I/O references.

3. An out-of-range memory reference will cause an interrupt condition to occur. For 034, 036, the interrupt will occur in 2 CP + 2 issues. For 035, 037, the interrupt will occur in 0 to 2 CP + 2 issues.

4. For 034, 036, memory reference out of limits will allow two parcels to issue. For 035, 037, two to four parcels will issue.

5. An uncorrected memory parity error will allow a minimum of 2 issues and a maximum of 7 CPs + 2 issues.

6. (Ai) = 0 causes a zero block transfer.

    200_8 > (Ai) > 100 causes a wrap-around condition

    (Ai) > 177_8, bits 2^7 through 2^{23} are truncated. The block transfer is equal to the value of 2^0 through 2^6.

7. (A_0) is used as the block length if i = 0.

```
+---------------------------------------------------------------+
|                                                               |
|     040ijkm    Transmit jkm to Si                             |
|                                                               |
|     041ijkm    Transmit complement of jkm to Si               |
|                                                               |
+---------------------------------------------------------------+
```

These two-parcel instructions provide for entering immediate values
into an S register.
The 040 instruction enters into Si a 64-bit value that is composed
of the 22-bit jkm field and 42 upper bits of zero.

The 041 instruction enters into Si a 64-bit value that is the complement
of a value formed by the 22-bit jkm field and 42 upper bits of zero. The
complement is formed by changing all one bits to zero and all zero bits
to one. Thus, for the 041 instruction, the upper 42 bits of Si are
set to one and the instruction provides for entering a negative value
into Si.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        S register access conflict
        Si reserved

Execution time
        Instruction issue
                Both parcels in same buffer   2 CPs
                Both parcels in different buffers   4 CPs
                Second parcel not in a buffer   13 CPs
        Si ready   1 CP

Special cases
        None

```
042ijk      Form 64 - jk bits of one's mask in Si from right
043ijk      Form jk bits of one's mask in Si from left
```

The 042 instruction generates a mask of 64-jk ones from right to left in Si. Thus, for example, if jk = 0, Si contains all one bits and if jk = 77_8, Si contains zeros in all but the lowest order bit.

The 043 instruction generates a mask of jk ones from left to right in Si. Thus, for example, if jk = 0, Si contains all zeroed bits and if jk = 77_8, Si contains ones in all but the lowest order bit.

These instructions are executed in the scalar logical unit.

Hold issue conditions

      034 - 037 in process

      Exchange in process

      S register access conflict

      Si reserved

Execution time

      Instruction issue 1 CP

      Si ready 1 CP

Special cases

      None

```
-----------------------------------------------------------------------
|                                                                     |
|   044ijk      Logical product of (Sj) and (Sk) to Si                |
|   045ijk      Logical product of (Sj) and complement of (Sk) to Si  |
|   046ijk      Logical difference of (Sj) and (Sk) to Si             |
|   047ijk      Logical equivalence of (Sk) and (Sj) to Si            |
|   050ijk      Scalar merge                                          |
|   051ijk      Logical sum of (Sj) and (Sk) to Si                    |
|                                                                     |
-----------------------------------------------------------------------
```

These instructions are executed in the scalar logical unit.

The 044 instruction forms the logical product (AND) of (Sj) and (Sk)
and enters the result into Si. Bits of Si are set to one when the
corresponding bits of (Sj) and (Sk) are one as in the following example:

$$(Sj) = 1\ 1\ 0\ 0$$
$$(Sk) = 1\ 0\ 1\ 0$$
$$\overline{}$$
$$(Si) = 1\ 0\ 0\ 0$$

(Sj) is transmitted to Si if the j and k designators have the same non-
zero value. Si is cleared if the j designator is zero. The sign bit
of (Sj) is extracted into Si if the j designator is nonzero and the k
designator is zero.

The 045 instruction forms the logical product (AND) of (Sj) and the
complement of (Sk) and enters the result into Si. Bits of Si are set
to one when the corresponding bits of (Sj) and the complement of (Sk)
are one as in the following example:

$$(Sj) = 1\ 1\ 0\ 0$$
$$(Sk) = 1\ 0\ 1\ 0$$
$$\overline{}$$
$$(Si) = 0\ 1\ 0\ 0$$

Si is cleared if the j and k designators have the same value or if the
j designator is zero. (Sj) with the sign bit cleared is transmitted
to Si if the j designator is non-zero and the k designator is zero.

The 046 instruction forms the logical difference (exclusive OR) of
(Sj) and (Sk) and enters the result into Si. Bits of Si are set to
one when the corresponding bits of (Sj) and (Sk) are different as in
the following example:

        (Sj) = 1 1 0 0
        (Sk) = 1 0 1 0
        (Si) = 0 1 1 0

Si is cleared if the j and k designators have the same nonzero value.
(Sk) is transmitted to Si if the j designator is zero and the k
designator is nonzero. The sign bit of (Sj) is complemented and the
result is transmitted to Si if the j designator is nonzero and the
k designator is zero.

The 047 instruction forms the logical equivalence of (Sj) and (Sk), and
enters the result into Si. Bits of Si are set to one when the
corresponding bits of (Sj) and (Sk) are the same as in the
following example:

        (Sj) = 1 1 0 0
        (Sk) = 1 0 1 0
        (Si) = 1 0 0 1

Si is set to all ones if the j and k designators have the same nonzero
value. The complement of (Sk) is transmitted to Si if the j designator
is zero and the k designator is nonzero. All bits except the sign bit
of (Sj) are complemented and the result is transmitted to Si if the j
designator is nonzero and the k designator is zero.

The 050 instruction merges the contents of (Sj) with (Si) depending
on the ones mask in Sk. The result is defined by the Boolean equation
(Si) = (Sj)(Sk) + (Si)($\overline{Sk}$) as illustrated in the following example:

        (Sk) = 1 1 1 1 0 0 0 0
        (Si) = 1 1 0 0 1 1 0 0
        (Sj) = 1 0 1 0 1 0 1 0
        (Si) = 1 0 1 0 1 1 0 0

The 050 instruction is intended for merging portions of 64-bit words into a composite word. Bits of Si are cleared when the corresponding bits of Sk are one if the j designator is zero and the k designator is nonzero. The sign bit of (Sj) replaces the sign bit of Si if the j designator is nonzero and the k designator is zero. The sign bit of Si is cleared if the j and k designators are both zero.

The 051 instruction forms the logical sum (inclusive OR) of (Sj) and (Sk) and enters the result into Si. Bits of Si are set when one of the corresponding bits of (Sj) and (Sk) is set as in the following example:

$$(Sj) = 1\ 1\ 0\ 0$$
$$(Sk) = \underline{1\ 0\ 1\ 0}$$
$$(Si) = 1\ 1\ 1\ 0$$

(Sj) is transmitted to Si if the j and k designators have the same nonzero value. (Sk) is transmitted to Si if the j designator is zero and the k designator is nonzero. (Sj) with the sign bit set to one is transmitted to Si if the j designator is nonzero and the k designator is zero. A ones mask consisting of only the sign bit is entered into Si if the j and k designators are both zero.

Hold issue conditions

    034 - 037 in process
    Exchange in process
    S register access conflict
    Si, Sj, and Sk reserved

Execution time

    Si ready 1 CP
    Instruction issue 1 CP

Special cases

    (Sj) = 0 if j = 0
    (Sk) = 2^{63} if k = 0

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                   │
│    052ijk      Shift (Si) left jk places to $S_0$                 │
│    053ijk      Shift (Si) right 64 - jk places to $S_0$           │
│    054ijk      Shift (Si) left jk places to Si                    │
│    055ijk      Shift (Si) right 64 - jk places to Si              │
│                                                                   │
└─────────────────────────────────────────────────────────────────┘
```

These instructions are executed in the scalar shift unit. They
shift values in an S register by an amount specified by jk. All
shifts are end off with zero fill.

The 052 instruction shifts (Si) left jk places and enters the result
into S_0.
The 053 instruction shifts (Si) right by 64-jk places and enters the
result into S_0.
The 054 instruction shifts (Si) left jk places and enters the result
into Si.
The 055 instruction shifts (Si) right by 64-jk places and enters the
result into Si.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        S register access conflict
        Si reserved
        S_0 reserved (052 and 053 only)

Execution time
        For 052, 053, S_0 ready 2 CPs
        For 054, 055, Si ready 2 CPs
        Instruction issue 1 CP

Special cases
        None

2240004   4-38   E

```
 ----------------------------------------------------------------------
|     056ijk    Shift (Si) and (Sj) left by (Ak) places to Si          |
|     057ijk    Shift (Sj) and (Si) right by (Ak) places to Si         |
 ----------------------------------------------------------------------
```

These instructions are executed in the scalar shift unit. They shift
128-bit values fromed by logically joining two S registers. Shift counts
are obtained from register Ak. A shift of one place occurs if the k
designator is zero.

All shifts are end-off with zero fill. The shift is effectively a
circular shift if the shift count does not exceed 64 and the i and j
designators are equal and nonzero. For both the 056 and 057 instructions,
(Sj) are unchanged.

The 056 instruction performs left shifts of (Si) and (Sj) with (Si)
initially the most significant bits of the double register. The high-
order 64 bits of the result are transmitted to Si. Si is cleared if the
shift count exceeds 127. The 056 instruction produces the same result
as the 054 instruction if the shift count does not exceed 63 and the j
designator is zero.

The 057 instruction performs right shifts of (Sj) and (Si) with (Sj)
initially the most significant bits of the double register. The low-order
64 bits of the result are transmitted to Si. Si is cleared if the shift
count exceeds 127. The 057 instruction produces the same result as the
055 instruction if the shift count does not exceed 63 and the j designator
is zero.

Hold issue conditions

    034 - 037 in process

    Exchange in process

    S register access conflict

    Si, Sj, or Ak reserved

Execution time

    Si ready   3 CPs

    Instruction issue   1 CP

Special cases

    (Sj) = 0 if j = 0

    (Ak) = 1 if k = 0

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                   │
│     060ijk      Integer sum of (Sj) and (Sk) to Si                │
│     061ijk      Integer difference of (Sj) and (Sk) to Si         │
│                                                                   │
└─────────────────────────────────────────────────────────────────┘
```

These instructions are executed in the scalar add unit.

The 060 instruction forms the integer sum of (Sj) and (Sk) and enters the result into Si. No overflow is detected.

The 061 instruction forms the integer difference of (Sj) and (Sk) and enters the result into Si. No overflow is detected.

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si, Sj, or Sk reserved

Execution time

Si ready   3 CPs

Instruction issue   1 CP

Special cases

For 060:

(Si) = (Sk)   if j = 0 and k ≠ 0

(Si) = 2^{63} if j = 0 and k = 0

(Si) = (Sj) with 2^{63} complemented if j ≠ 0 and k = 0

For 061:

(Si) = -(Sk)   if j = 0 and k ≠ 0

(Si) = (Sj) with 2^{63} complemented if j ≠ 0 and k = 0

```
--------------------------------------------------------------------------
|                                                                        |
|    062ijk     Floating sum of (Sj) and (Sk) to Si                      |
|    063ijk     Floating difference of (Sj) and (Sk) to Si               |
|                                                                        |
--------------------------------------------------------------------------
```

These instructions are performed by the floating point add unit. Operands are assumed to be in floating point format. The result is normalized even if the operands are unnormalized. Underflow and overflow conditions are described in Section 3.

The 062 instruction forms the sum of the floating point quantities in Sj and Sk and enters the normalized result into Si.

The 063 instruction forms the difference of the floating point quantities in Sj and Sk and enters the normalized result into Si.

Hold issue conditions

 034 - 037 in process
 Exchange in process
 Si register access conflict
 Si, Sj, or Sk reserved
 170 - 173 in process; unit busy (VL) + 4 CPs

Execution time

 Si ready 6 CPs
 Instruction issue 1 CP

Special cases

For 062:

 (Si) = (Sk) normalized if j = 0 and k ≠ 0
 (Si) = (Sj) normalized if (Sj) exponent is valid, j ≠ 0 and k = 0

For 063:

 (Si) = -(Sk) normalized if j = 0 and k ≠ 0
 (Si) = (Sj) normalized if (Sj) exponent is valid, j ≠ 0 and k = 0

 Arithmetic error allows 0 to 9 CPs + 2 parcels to issue before interrupt occurs if f.p. error flag is set.

```
064ijk    Floating product of (Sj) and (Sk) to Si
065ijk    Half-precision rounded floating product of (Sj) and
          (Sk) to Si
066ijk    Rounded floating product of (Sj) and (Sk) to Si
067ijk    Reciprocal iteration; 2 - (Sj) * (Sk) to Si
```

These instructions are executed by the floating point multiply unit.
Operands are assumed to be in floating point format. The result is
not guaranteed to be normalized if the operands are unnormalized.

The 064 instruction forms the product of the floating point quantities
in Sj and Sk and enters the result into Si.

The 065 instruction forms the half-precision rounded product of the
floating point quantities in Sj and Sk and enters the result into Si.
The low order 18 bits of the result are cleared.

The 066 instruction forms the rounded product of the floating point
quantities in Sj and Sk and enters the result into Si.

The 067 instruction forms two minus the product of the floating point
quantities in Sj and Sk and enters the result into Si. This instruction
is used in the divide sequence as described in Section 3 under Floating
Point Arithmetic.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        S register access conflict
        Si, Sj, or Sk reserved
        160 - 167 in process; unit busy (VL) + 4 CPs

Execution time

    Instruction issue 1 CP

    Si ready 7 CPs

Special cases

    $(Sj) = 0$ if $j = 0$

    $(Sk) = 2^{63}$ if $k = 0$

    Arithmetic error allows 9 CP + 2 parcels to issue before interrupt occurs if f.p. error flag is set.

```
:----------------------------------------------------------------------:
|                                                                        |
|    070ijx     Floating reciprocal approximation of (Sj) to Si         |
|                                                                        |
:----------------------------------------------------------------------:
```

This instruction is executed in the reciprocal approximation unit.

The instruction forms an approximation to the reciprocal of the normalized floating point quantity in Sj and enters the result into Si. This instruction occurs in the divide sequence to compute the quotient of two floating point quantities as described in Section 3 under Floating Point Arithmetic.

The reciprocal approximation instruction produces a result that is accurate to 30 bits. A second approximation may be generated to extend the accuracy to 47 bits using the reciprocal iteration instruction.

Hold issue conditions

        034 - 037 in process
        Exchange in process
        Si or Sj reserved
        174 in process; unit busy (VL) + 4 CPs

Execution time

        Si ready 14 CPs
        Instruction issue 1 CP

Special cases

        An arithmetic error allows 17 CPs + 2 parcels to issue if the
        f.p. error flag is set.

        (Si) is meaningless if (Sj) is not normalized; the unit assumes
        that bit 2^{47} of (Sj) = 1; no test is made of this bit.

        (Sj) = 0 produces a range error; the result is meaningless.

        (Sj) = 0 if j = 0.

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                   │
│    071ijk     Transmit (Ak) or normalized floating point constant │
│               to Si                                               │
│                                                                   │
└─────────────────────────────────────────────────────────────────┘
```

This instruction performs one of several functions depending on the
value of the j designator. The functions are concerned with trans-
mitting information from an A register to an S register and with
generating frequently used floating point constants.

071i0k	Transmit (Ak) to Si with no sign extension
071i1k	Transmit (Ak) to Si with sign extension
071i2k	Transmit (Ak) to Si as unnormalized floating point number
071i3k	Transmit constant 0.75×2^{48} to Si
071i4k	Transmit constant 0.5 to Si
071i5k	Transmit constant 1.0 to Si
071i6k	Transmit constant 2.0 to Si
071i7k	Transmit constant 4.0 to Si

When the j designator is 0, the 24-bit value in Ak is transmitted to
Si. The value is treated as an unsigned integer. The high-order bits
of Si are cleared.

When the j designator is 1, the 24-bit value in Ak is transmitted to
Si. The value is treated as a signed integer. The sign bit of Ak is
extended to the high order bit of Si.

When the j designator is 2, the 24-bit value in Ak is transmitted to
Si as an unnormalized floating point quantity. The result can then
be added to zero to normalize. For this instruction, the exponent in
bits 1 through 15 is set to 40060_8. The sign of the coefficient is
set according to the sign of Ak. If the sign bit of Ak is set, the
two's complement of Ak is entered into Si as the magnitude of the
coefficient and bit 0 of Si is set for the sign of the coefficient.

When the j designator is 3, the constant 0.75×2^{48} is entered into Si.

When the j designator is 4, 5, 6, or 7, the normalized floating point constant 0.5, 1.0, 2.0, or 4.0, respectively is transmitted to Si.

Hold issue conditions

034 - 037 in process

Exchange in process

Si register access conflict

Si reserved

Ak reserved (all instructions)

Execution time

Si ready 2 CPs
Instruction issue 1 CP

Special cases

$(A_0) = 1$ if $k = 0$

$(Si) = (Ak)$ if $j = 0$

$(Si) = (Ak)$ sign extended if $j = 1$

$(Si) = (Ak)$ unnormalized if $j = 2$

$(Si) = 0.6 \times 2^{60}$ (octal) if $j = 3$

$(Si) = 0.4 \times 2^{0}$ (octal) if $j = 4$

$(Si) = 0.4 \times 2^{1}$ (octal) if $j = 5$

$(Si) = 0.4 \times 2^{2}$ (octal) if $j = 6$

$(Si) = 0.4 \times 2^{3}$ (octal) if $j = 7$

```
072ixx     Transmit (RTC) to Si
073ixx     Transmit (VM) to Si
074ijk     Transmit (Tjk) to Si
075ijk     Transmit (Si) to Tjk
```

These instructions transmit register values to Si except for instruction
075 which transmits (Si) to Tjk.

The 072 instruction enters the 64-bit value of the real-time clock into
Si. The clock is incremented by one each clock period. The real-time
clock is cleared by the operating system at system initialization and
can be reset only by the monitor through use of the 0014 instruction.

The 073 instruction enters the 64-bit value of the vector mask (VM)
register into Si. The VM register is usually read after having been set
by the 175 instruction.

The 074 instruction enters the contents of Tjk into Si.

The 075 instruction enters the contents of Si into Tjk.

Hold issue conditions

    034 - 037 in process
    Exchange in process
    Si register access conflict (072, 073, and 074 only)
    Si reserved

For 073 only:

    175 in process, unit busy (VL) + 6 CPs
    003 in process, unit busy 6 CPs

Execution time

    Instruction issue 1 CP
    For 072 through 074, Si ready 1 CP
    For 075, Tjk ready 1 CP

Special cases

    None

```
: ----------------------------------------------------------------------- :
:    076ijk      Transmit (Vj element (Ak)) to Si                          :
:    077ijk      Transmit (Sj) to Vi element (Ak)                          :
: ----------------------------------------------------------------------- :
```

These instructions transmit a 64-bit quantity between a V register element and an S register.

The 076 instruction transmits the contents of an element of register Vj to Si.

The 077 instruction transmits the contents of register Sj to an element of register Vi.

The low-order six bits of (Ak) determine the vector element for either instruction.

Hold issue conditions

        034 - 037 in process
        Exchange in process
        Ak reserved
        Si register access conflict (076 only)
        For 076, Si and Vj reserved
        For 077, Vi and Sj reserved

Execution time

        Instruction issue 1 CP
        For 076, Si ready 5 CPs
        For 077, Vi ready 3 CPs

Special cases

        (Sj) = 0 if j = 0
        (Ak) = 1 if k = 0

```
  10hijkm     Read from ((Ah) + jkm) to Ai
  11hijkm     Store (Ai) to (Ah) + jkm
  12hijkm     Read from ((Ah) + jkm) to Si
  13hijkm     Store (Si) to (Ah) + jkm
```

These two parcel instructions transmit data between memory and an A
register or an S register. The content of Ah is added to the signed
integer in the jkm field to determine the memory address. If h is 0,
(Ah) is 0 and only the jkm field is used for the address. The address
arithmetic is performed by an address adder similar to but separate
from the address add unit.

The 10h and 11h instructions transmit 24-bit quantities to or from
A registers. When transmitting data from memory to an A register, the
upper 40 bits of the memory word are ignored. On a store from Ai into
memory, the upper 40 bits of the memory word are zeroed.

The 12h and 13h instructions transmit 64-bit quantities to or from
register Si.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        Rank A conflict and unit busy 3 CPs
        Rank B conflict and unit busy 2 CPs
        Rank C conflict and unit busy 1 CP
        Storage hold continuation
        Ah reserved
        For 10h and 11h only, Ai reserved
        For 12h and 13h only, Si reserved
        For 12h only, Si register access conflict
        Fetch request in last clock period

Execution time

Instruction issue:

Both parcels in same buffer   2 CPs

Parcels in different buffers   4 CPs

Second parcel not in a buffer   13 CPs

10h only, Ai ready   11 CPs

12h only, Si ready   11 CPs

Memory ready for next scalar read or store   4 CPs

Special cases

Rank A conflict, 3 CPs delay before Si ready

Rank B conflict, 2 CPs delay before Si ready

Rank C conflict, 1 CP delay before Si ready

Hold storage, 1 CP delay if 070 access conflict occurs

An uncorrected memory parity error will allow 14 CP + 2 parcels to issue

An out of range error will allow 5 CP + 2 parcels to issue

(Ah) = 0 if h = 0

```
140ijk    Logical products of (Sj) and (Vk elements) to Vi
          elements
141ijk    Logical products of (Vj elements) and (Vk elements)
          to Vi elements
142ijk    Logical sums of (Sj) and (Vk elements) to Vi elements
143ijk    Logical sums of (Vj elements) and (Vk elements) to
          Vi elements
144ijk    Logical differences of (Sj) and (Vk elements) to
          Vi elements
145ijk    Logical differences of (Vj elements) and (Vk elements)
          to Vi elements
146ijk    If VM bit = 1, transmit (Sj) to Vi elements
          If VM bit = 0, transmit (Vk elements) to Vi elements
147ijk    If VM bit = 1, transmit (Vj elements) to Vi elements
          If VM bit = 0, transmit (Vk elements) to Vi elements
```

These instructions are executed by the vector logical unit. The number
of operations performed is determined by the contents of the VL register.
All operations start with element zero of the Vi, Vj, or Vk register and
increment the element number by one for each operation performed. All
results are delivered to Vi.

For instructions 140, 142, 144, and 146, the content of Sj is delivered
to the functional unit for each operation as one of the operands. For
instructions 141, 143, 145, and 147, all operands are obtained from V
registers.

Instructions 140 and 141 form the logical products (AND) of pairs of
operands and enter the result into Vi. Bits of an element of Vi are set
to one when the corresponding bits of (Sj) or (Vj element) and (Vk
element) are one as in the following:

```
(Sj) or (Vj element) = 1 1 0 0
(Vk element)         = 1 0 1 0
(Vi element)         = 1 0 0 0
```

The 142 and 143 instructions form the logical sums (inclusive OR) of pairs of operands and deliver the results to Vi. Bits of an element of Vi are set to one when one of the corresponding bits of (Sj) or (Vj element) and (Vk element) is one as in the following:

```
(Sj) or (Vj element) = 1 1 0 0
(Vk element)         = 1 0 1 0
(Vi element)         = 1 1 1 0
```

The 144 and 145 instructions form the logical differences (exclusive OR) of pairs of operands and deliver the results to Vi. Bits of an element are set to one when the corresponding bit of (Sj) or (Vj element) are different from (Vk element) as in the following:

```
(Sj) or (Vj element) = 1 1 0 0
(Vk element)         = 1 0 1 0
(Vi element)         = 0 1 1 0
```

The 146 and 147 instructions transmit operands to Vi depending on the contents of the vector mask register (VM). Bit 0 of the mask corresponds to element 0 of a V register. Bit 63 corresponds to element 63. Operand pairs used for the selection depend on the instruction. For the 146 instructions, the first operand is always (Sj), the second operand is (Vk element). For the 147 instruction, the first operand is (Vj element) and the second operand is (Vk element). If bit n of the vector mask is one, the first operand is transmitted; if bit n of the mask is zero, the second operand (Vk element) is selected.

Examples

1. Suppose that a 146 instruction is to be executed and the following register conditions exist:

    (VL) = 4
    (VM) = 0 60000 0000 0000 0000 0000
    (S2) = -1
    (Element 0) of V6 = 1
    (Element 1) of V6 = 2
    (Element 2) of V6 = 3
    (Element 3) of V6 = 4

    Instruction 146726 is executed and following execution, the first four elements of V7 contain the following values:

    (Element 0) of V7 = 1
    (Element 1) of V7 = -1
    (Element 2) of V7 = -1
    (Element 3) of V7 = 4

    The remaining elements of V7 are unaltered.

2. Suppose that a 147 instruction is to be executed and the following register conditions exist:

    (VL) = 4
    (VM) = 0 600000 0000 0000 0000 0000

    (Element 0) of V2 = 1   (Element 0) of V3 = -1
    (Element 1) of V2 = 2   (Element 1) of V3 = -2
    (Element 2) of V3 = 3   (Element 2) of V3 = -3
    (Element 3) of V4 = 4   (Element 3) of V3 = -4

    Instruction 147123 is executed and following execution, the first four elements of V1 contain the following values:

    (Element 0) of V1 = -1
    (Element 1) of V1 = 2
    (Element 2) of V1 = 3
    (Element 3) of V1 = -4

    The remaining elements of V1 are unaltered.

Hold issue conditions

    034 - 037 in process
    Exchange in process
    Vi or Vk reserved
    14x in process, unit busy (VL) + 4 CPs
    175 in process, unit busy (VL) + 4 CPs
    003 in process, unit busy 3 CPs
    For 140, 142, 144, 146 only, Sj reserved
    For 141, 143, 145, 147 only, Vj reserved

Execution time

    Instruction issue 1 CP
    Vi ready   9 CPs if (VL) $\leq$ 5
    Vi ready   (VL) + 4 CPs if (VL) > 5
    Vj or Vk ready 5 CPs if (VL) $\leq$ 5
    Vj or Vk ready (VL) CPs if (VL) > 5
    Unit ready (VL) + 4 CPs
    Chain slot ready 4 CPs

Special cases

    (Sj) = 0 if j = 0

```
┌─────────────────────────────────────────────────────────────────────┐
│                                                                       │
│   150ijk     Single shift of (Vj elements) left by (Ak) places to     │
│              Vi elements                                              │
│                                                                       │
│   151ijk     Single shift of (Vj elements) right by (Ak) places to    │
│              Vi elements                                              │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘
```

These instructions are executed in the vector shift unit. The number
of operations performed is determined by the contents of the VL register.
Operations start with element 0 of the Vi and Vj registers and end with
elements specified by the contents of VL-1.

All shifts are end-off with zero fill. The shift count is obtained
from (Ak) and elements of Vi are cleared if the shift count exceeds 63.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        Vi or Vj reserved
        Ak reserved
        150 - 153 in process, unit busy (VL) + 4 CPs

Execution time
        Instruction issue 1 CP
        Vi ready 11 CPs if (VL) $\leq$ 5
        Vi ready (VL) + 6 CPs if (VL) > 5
        Vj ready 5 CPs if (VL) $\leq$ 5
        Vj ready (VL) CPs if (VL) > 5
        Unit ready (VL) + 4 CPs
        Chain slot ready 6 CPs

Special cases
        (Ak) = 1 if k = 0

```
--------------------------------------------------------------------------------
  152ijk       Double shifts of (Vj elements) left (Ak) places to
               Vi elements

  153ijk       Double shifts of (Vj elements) right (Ak) places to
               Vi elements
--------------------------------------------------------------------------------
```

These instructions are executed in the vector shift unit. They shift
128-bit values formed by logically joining the contents of two elements
of the Vj register. The direction of the shift determines whether the
upper bits or the lower bits of the result are sent to Vi. Shift counts
are obtained from register Ak.

All shifts are end-off with zero fill.

The number of operations is determined by the contents of the VL register.

The 152 instruction performs left shifts. In the general case, element
0 of Vj is joined with element 1 and the 128-bit quantity is shifted left
by the amount specified by (Ak). The 64 high order bits of the result
are transmitted to element 0 of Vi. The figure below illustrates this
operation.

```
           +-----------------------+-----------------------+
           |   (Element 0) of Vj   |   (Element 1) of Vj   |
           +-----------------------+-----------------------+
  +-----------------------+-----------------------+
  | (Element 0) of Vj     | (Element 1) of Vj     | ◄── (Ak)
  +-----------------------+-----------------------+
  _____/  _____/
     End off            64-bit result to element 0 of Vi
```
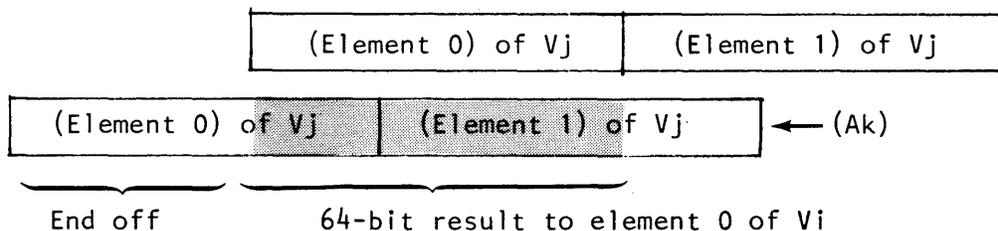
If (VL) were 1, element 0 would have been joined with 64 bits of zero and
only the one operation would be performed. If (VL) > 2, the operation
continues by joining element 1 with element 2 and transmitting the 64-bit
result to element 1 of Vi. This is illustrated as follows:

```
           +-----------------------+-----------------------+
           |   (Element 1) of Vj   |   (Element 2) of Vj   |
           +-----------------------+-----------------------+
  +-----------------------+-----------------------+
  | (Element 1) of Vj     | (Element 2) of Vj     | ◄── (Ak)
  +-----------------------+-----------------------+
  _____/  _____/
     End off            64-bit result to element 1 of Vi
```
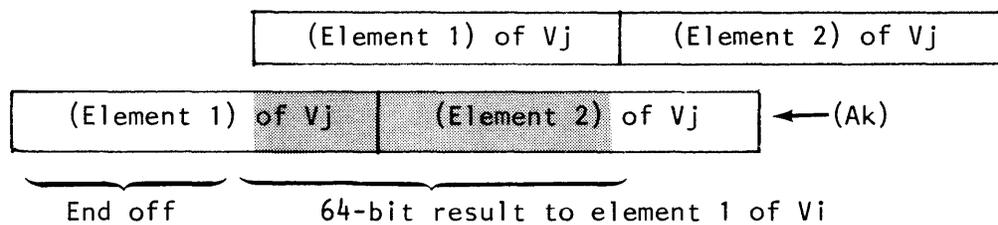
If (VL) were 2, however, element 1 would have been joined with 64 bits of zero and only two operations would be performed. Thus, the last element of Vj as determined by (VL) is joined with 64 bits of zeros. The following figure illustrates this operation.
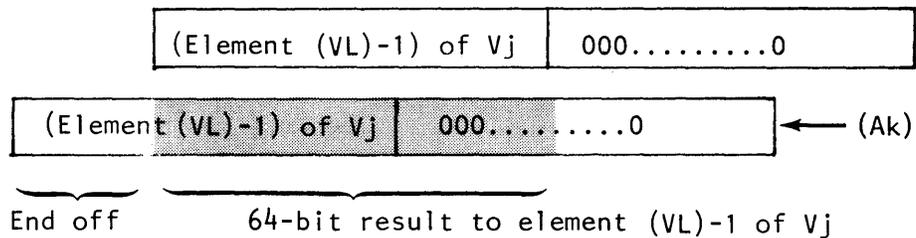

```
          ┌─────────────────────────┬──────────────────┐
          │ (Element (VL)-1) of Vj  │ 000.........0    │
          └─────────────────────────┴──────────────────┘
     ┌─────────────────────────┬──────────────────────────┐
     │ (Element (VL)-1) of Vj  │ 000.........0            │ ◄── (Ak)
     └─────────────────────────┴──────────────────────────┘
     _____v_____/ _____v_____/
         End off          64-bit result to element (VL)-1 of Vj
```

If (Ak) > 128, the result is all zeros. If (Ak) > 64, the result register contains (Ak) - 64 zeros.

Example:

Suppose that a 152 instruction is to be executed and the following register conditions exist:

        (VL) = 4
        (A1) = 3
        (Element 0) of V4 = 0 00000 0000 0000 0000 0007
        (Element 1) of V4 = 0 60000 0000 0000 0000 0005
        (Element 2) of V4 = 1 00000 0000 0000 0000 0006
        (Element 3) of V4 = 1 60000 0000 0000 0000 0007

Instruction 152541 is executed and following execution, the first four elements of V5 contain the following values:

        (Element 0) of V5 = 0 00000 0000 0000 0000 0073
        (Element 1) of V5 = 0 00000 0000 0000 0000 0054
        (Element 2) of V5 = 0 00000 0000 0000 0000 0067
        (Element 3) of V5 = 0 00000 0000 0000 0000 0070

The 153 instruction performs right shifts. Element 0 of Vj is joined with 64 low-order bits of zero and the 128 bit quantity is shifted right by the amount specified by (Ak). The 64 low-order bits of the result are transmitted to element 0 of Vi. The figure below illustrates this operation.

```
┌────────────────────┬────────────────────┐
│ 000........0        │ (Element 0) of Vj  │
└────────────────────┴────────────────────┘

         ┌────────────────────┬────────────────────┐
(Ak) ──▶ │ 000........0        │ (Element 0) of Vj  │
         └────────────────────┴────────────────────┘
          ╰─────────────────────────────╯ ╰────────╯
              64-bit result to              End off
              element 0 of Vi
```
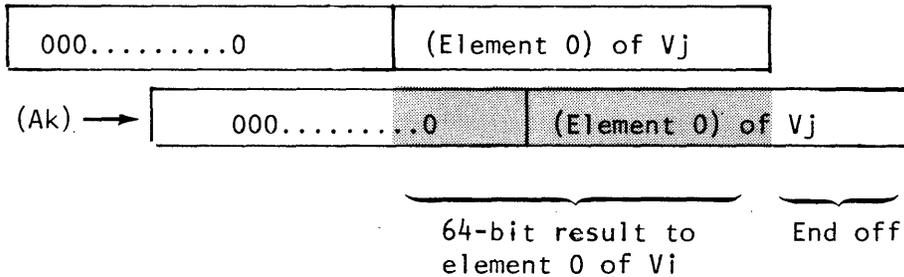
If (VL) = 1, only the one operation is performed. In the general case, however, instruction execution continues by joining element 0 with element 1, shifting the 128-bit quantity by the amount specified by (Ak), and transmitting the result to element 1 of Vi. This operation is shown below.

```
┌────────────────────┬────────────────────┐
│ (Element 0) of Vj  │ (Element 1) of Vj  │
└────────────────────┴────────────────────┘

         ┌────────────────────┬────────────────────┐
(Ak) ──▶ │ (Element 0) of Vj  │ (Element 1) of Vj  │
         └────────────────────┴────────────────────┘
          ╰─────────────────────────────╯ ╰────────╯
              64-bit result to              End off
              element 1 of Vj
```
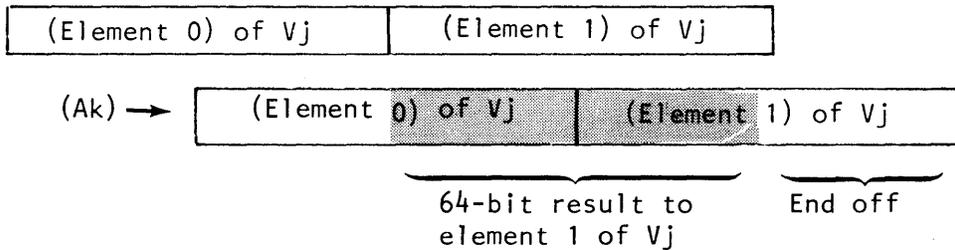
The last operation performed by the instruction joins the last element of Vj as determined by (VL) with the preceding element. The following figure illustrates this operation.
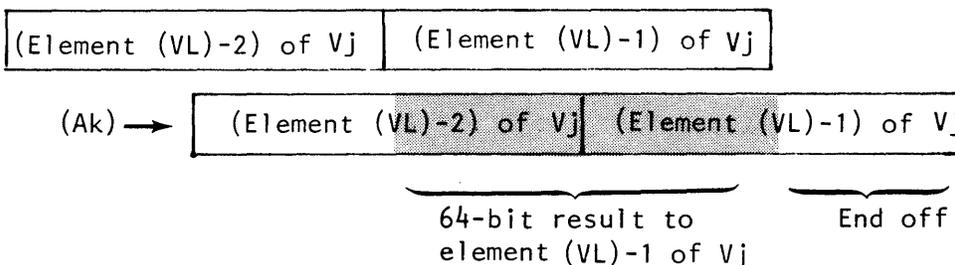
```
┌────────────────────────┬────────────────────────┐
│ (Element (VL)-2) of Vj │ (Element (VL)-1) of Vj │
└────────────────────────┴────────────────────────┘

         ┌────────────────────────┬────────────────────────┐
(Ak) ──▶ │ (Element (VL)-2) of Vj │ (Element (VL)-1) of Vj │
         └────────────────────────┴────────────────────────┘
          ╰───────────────────────────────────╯ ╰──────────╯
              64-bit result to                     End off
              element (VL)-1 of Vj
```

If (Ak) > 128, the result is all zeros. If (Ak) > 64, the result register contains (Ak) - 64 zeros.

Example:

Suppose that a 153 instruction is to be executed and the following register conditions exist:

> (VL) = 4
> (A6) = 3
> (Element 0) of V_2 = 0 00000 0000 0000 0000 0017
> (Element 1) of V_2 = 0 60000 0000 0000 0000 0006
> (Element 2) of V_2 = 1 00000 0000 0000 0000 0006
> (Element 3) of V_2 = 1 60000 0000 0000 0000 0007

Instruction 153026 is executed and following execution, register V_0 contains the following values:

> (Element 0) of V_0 = 0 00000 0000 0000 0000 0001
> (Element 1) of V_0 = 1 66000 0000 0000 0000 0000
> (Element 2) of V_0 = 1 50000 0000 0000 0000 0000
> (Element 3) of V_0 = 1 56000 0000 0000 0000 0000

The remaining elements of V_0 are unaltered.

Hold issue conditions

> 034 - 037 in process
> Exchange in process
> Vi or Vj reserved
> Ak reserved
> 150 - 153 in process, unit busy (VL) + 4 CPs

Execution time

> Instruction issue 1 CP
> Vi ready 11 CPs if (VL) $\leq$ 5
> Vi ready (VL) + 6 CPs if (VL) > 5

Execution time (continued)
$$Vj \text{ ready } 5 \text{ CPs if } (VL) \leq 5$$
$$Vj \text{ ready } (VL) \text{ CPs if } (VL) > 5$$
$$Unit \text{ ready } (VL) + 4 \text{ CPs}$$
$$Chain \text{ slot ready } 6 \text{ CPs}$$

Special cases
$$(Ak) = 1 \text{ if } k = 0$$

```
--------------------------------------------------------------------------
|                                                                        |
|   154ijk      Integer sums of (Sj) and (Vk elements) to Vi elements    |
|   155ijk      Integer sums of (Vj elements) and (Vk elements) to       |
|                 Vi elements                                            |
|   156ijk      Integer differences of (Sj) and (Vk elements) to         |
|                 Vi elements                                            |
|   157ijk      Integer differences of (Vj elements) and (Vk elements)   |
|                 to Vi elements                                         |
|                                                                        |
--------------------------------------------------------------------------
```

These instructions are executed by the vector add unit.

Instructions 154 and 156 perform integer addition. Instructions 155
and 157 perform integer subtraction. The number of additions or
subtractions performed is determined by the contents of the VL register.
All operations start with element zero of the V registers and increment
the element number by one for each operation performed. All results
are delivered to elements of Vi. No overflow is detected.

Instructions 154 and 156 deliver (Sj) to the functional unit as one
of the operands for each operation. The other operand is an element
of Vk. For instructions 155 and 157, both operands are obtained from
V registers.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        Vi or Vk reserved
        154 - 157 in process, unit busy (VL) + 4 CPs
        For 154 and 156 only, Sj reserved
        For 155 and 157 only, Vj reserved

Execution time

    Instruction issue 1 CP

    Vi ready 10 CPs if (VL) $\leq$ 5

    Vi ready (VL) + 5 CPs if (VL) > 5

    Vj or Vk ready 5 CPs if (VL) $\leq$ 5

    Vj or Vk ready (VL) CPs if (VL) > 5

    Unit ready (VL) + 4 CPs

    Chain slot ready 5 CPs

Special cases

   For 154, if j = 0, then (Sj) = 0 and (Vi element) = (Vk element)

    For 156, if j = 0, then (Sj) = 0 and (Vi element) = -(Vk element)

```
160ijk      Floating products of (Sj) and (Vk elements) to Vi
            elements
161ijk      Floating products of (Vj elements) and (Vk elements)
            to Vi elements
162ijk      Half-precision rounded floating products of (Sj)
            and (Vk elements) to Vi elements
163ijk      Half-precision rounded floating products of (Vj
            elements) and (Vk elements) to Vi elements
164ijk      Rounded floating products of (Sj) and (Vk elements)
            to Vi elements
165ijk      Rounded floating products of (Vj elements) and (Vk
            elements) to Vi elements
166ijk      Reciprocal iterations; 2 - (Sj) * (Vk elements) to
            Vi elements
167ijk      Reciprocal iterations; 2 - (Vj elements) * (Vk
            elements) to Vi elements
```

These instructions are executed in the floating point multiply unit.
The number of operations performed by an instruction is determined by
the contents of the VL register. All operations start with element
zero of the V registers and increment the element number by one for
each success operation.

Operands are assumed to be in floating point format. Even-numbered
instructions in the group deliver (Sj) to the functional unit for each
operation as one of the operands. The other operand is an element of
Vk. For odd-numbered instructions in the group, both operands are
obtained from V registers.

All results are delivered to elements of Vi. If the operands are
unnormalized, there is no guarantee that the products will be normalized.

Out of range conditions are described in Section 3.

The 160 instruction forms the products of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the results into Vi.

The 161 instruction forms the products of the floating point quantities in elements of Vj and Vk and enters the results into Vi.

The 162 instruction forms the half-precision rounded products of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the results into Vi. The low order 18 bits of the result elements are zeroed.

The 163 instruction forms the half-precision rounded products of the floating point quantities in elements of Vj and Vk and enters the results into Vi. The low order 18 bits of the result elements are zeroed.

The 164 instruction forms the rounded products of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the results into Vi.

The 165 instruction forms the rounded products of the floating point quantities in elements of Vj and Vk and enters the results into Vi.

The 166 instruction forms for each element, two minus the product of the floating point quantity in Sj and the floating point quantity in elements of Vk. It then enters the results into Vi.

The 167 instruction forms for each element pair, two minus the product of the floating point quantities in elements of Vj and Vk and enters the results into Vi.

Hold issue conditions

034 - 037 in process

Exchange in process

Vi or Vk reserved

16x in process, unit busy (VL) + 4 CPs

For 160, 162, 164, and 166:

Sj reserved

For 161, 163, 165, and 167:

Vj reserved

Execution time

Instruction issue 1 CP

Vi ready 14 CPs if $(VL) \leq 5$

Vi ready (VL) + 9 CPs if $(VL) > 5$

Vj or Vk ready 5 CPS if $(VL) \leq 5$

Vj or Vk ready (VL) CPs if $(VL) > 5$

Unit ready (VL) + 4 CPs

Chain slot ready 9 CPs

Special cases

$(Sj) = 0$ if $j = 0$

Arithmetic error allows a minimum of 21 CP + 2 parcels
and a maximum of (VL) + 20 CP + 2 parcels to issue
before interrupt occurs if floating point error flag set.

```
-----------------------------------------------------------------------
|                                                                      |
|   170ijk      Floating sums of (Sj) and (Vk elements) to Vi          |
|               elements                                               |
|                                                                      |
|   171ijk      Floating sums of (Vj elements) and (Vk elements) to    |
|               Vi elements                                            |
|                                                                      |
|   172ijk      Floating differences of (Sj) and (Vk elements) to      |
|               Vi elements                                            |
|                                                                      |
|   173ijk      Floating differences of (Vj elements) and (Vk          |
|               elements) to Vi elements                               |
|                                                                      |
-----------------------------------------------------------------------
```

These instructions are executed by the floating point add unit.
Instructions 170 and 171 perform floating point addition; instructions
172 and 173 perform floating point subtraction. The number of additions
or subtractions performed by an instruction is determined by the contents
of the VL register. All operations start with element zero of the V
registers and increment the element number by one for each operation
performed. All results are delivered to Vi. The results are normalized
even if the operands are unnormalized.

Instructions 170 and 172 deliver (Sj) to the functional unit for each
operation as one of the operands. The other operand is an element of
Vk. For instructions 171 and 173, both operands are obtained from V
registers.

Out of range conditions are described in Section 3.

Hold issue conditions
        034 - 037 in process
        Exchange in process
        Vi or Vk reserved
        170 - 173 in process, unit busy (VL) + 4 CPs
        For 170, 172:
            Sj reserved
        For 171, 173:
            Vj reserved

<u>Execution time</u>

      Instruction issue 1 CP

      Vi ready 13 CPs if (VL) $\leq$ 5

      Vi ready (VL) + 8 CPs if (VL) > 5

      Vj and Vk ready 5 CPs if (VL) $\leq$ 5

      Vj and Vk ready (VL) CPs if (VL) > 5

      Unit ready (VL) + 4 CPs

      Chain slot ready 8 CPs

<u>Special cases</u>

      (Sj) = 0 if j = 0

      Arithmetic error allows a minimum of 13 CP + 2 parcels and
      a maximum of (VL) + 12 CP + 2 parcels to issue before
      interrupt occurs if f.p. error flag set.

```
-------------------------------------------------------------------------
:                                                                         :
:    174ij0      Floating point reciprocal approximations of (Vj          :
:                                                                         :
:                elements) to Vi elements                                 :
:                                                                         :
-------------------------------------------------------------------------
```

This instruction is executed in the reciprocal approximation unit.

The instruction forms an approximate value of the reciprocal of the
normalized floating point quantity in each element of Vj and enters
the result into elements of Vi. The number of elements for which
approximations are found is determined by the contents of the VL
register.

The 174 instruction occurs in the divide sequence to compute the
quotients of floating point quantities as described in Section 3
under Floating Point Arithmetic.

The reciprocal approximation instruction produces results that are
accurate to 30 bits. A second approximation may be generated to
extend the accuracy to 47 bits using the reciprocal iteration
instruction.

Hold issue conditions

        034 - 037 in process
        Exchange in process
        Vi or Vj reserved
        174 in process, unit busy for (VL) + 4 CPs

Execution time

        Instruction issue 1 CP
        Vi ready   21 CPs if (VL) $\leq$ 5
        Vi ready   (VL) + 16 CPs if (VL) > 5
        Vj ready   5 CPs if (VL) $\leq$ 5
        Vj ready   (VL) CPs if (VL) > 5
        Unit ready   (VL) + 4 CPs
        Chain slot ready   16 CPs

Special cases

(Vi element) is meaningless if (Vj element) is not normalized; the unit assumes that bit 2^{47} of (Vj element) is one; no test of this bit is made.

Arithmetic error allows a minimum of 21 CP + 2 parcels and a maximum of (VL) + 20 CP + 2 parcels to issue before interrupt occurs if f.p. error flag set.

If the Vector Population Instructions Option is installed, the k field becomes relevant and allows recognition of the 174ij1 and 174ij2 instructions. When this option is installed, the k field must be 0 for the floating point reciprocal approximation instruction.

```
--------------------------------------------------------------------------------
|   174ij1    Population counts of (Vj elements) to Vi elements              |
|   174ij2    Population count parities of (Vj elements) to Vi elements      |
--------------------------------------------------------------------------------
```

These instructions require the presence of the Vector Population Instruc-
tions Option. If this option is not installed, these instructions are
executed as vector reciprocal approximation instructions.

The 174ij1 instruction counts the number of bits set to one in each
element of Vj and enters the results into corresponding elements of Vi.
The results are entered into the low order 7 bits of each Vi element;
the remaining higher order bits of each Vi element are zeroed.

The 174ij2 instruction counts the number of bits set to one in each
element of Vj. The least significant bit of each element result shows
whether the result is an odd or even number. Only the least significant
bit of each element is transferred to the least significant bit position
of the corresponding element of register Vi. The actual population count
results are not transferred.
These instructions are implemented in the vector population count functional
unit which requires the presence of the Vector Population Instructions Option.

Hold issue conditions

    034-037 in process
    Exchange in process
    Vi reserved
    Vk reserved
    174 in process; unit busy for (VL) + 4 CPs

Execution time

    Instruction issue 1 CP
    Vi ready   13 CPS if (VL) $\leq$ 5
    Vi ready   (VL) + 8CPs if (VL) > 5
    Vj ready   5 CPs if (VL) $\leq$ 5
    Vj ready   (VL) CPs if (VL) > 5
    Unit ready (VL) + 4 CPs
    Chain slot ready 8 CPs

```
┌─────────────────────────────────────────────────────────────────────┐
│     175xjk    Test (Vj elements) and enter test results              │
│               into VM; the type of test made is defined by k         │
└─────────────────────────────────────────────────────────────────────┘
```

This instruction creates a vector mask in VM based on the results of
testing the contents of the elements of register Vj. Each bit of VM
corresponds to an element of Vj. Bit 0 corresponds to element 0;
bit 63 corresponds to element 63.

The type of test made by the instruction depends on the lower two bits
of the k designator. The upper bit of the k designator is not
interpreted.
If the k designator is 0, the VM bit is set to one when (Vj element)
is zero and is set to zero when (Vj element) is nonzero.

If the k designator is 1, the VM bit is set to one when (Vj element)
is nonzero and is set to zero when (Vj element) is zero.

If the k designator is 2, the VM bit is set to one when (Vj element)
is positive and is set to zero when (Vj element) is negative. A zero
value is considered positive.

If the k designator is 3, the VM bit is set to one when (Vj element)
is negative and is set to zero when (Vj element) is positive. A zero
value is considered positive.

The number of elements tested is determined by the contents of the VL
register. VM bits corresponding to untested elements of Vj are zeroed.

The 175 vector mask instruction provides a vector counterpart to the
scalar conditional branch instructions.

The 175 vector mask instruction uses the vector logical unit.

Hold issue conditions

      034 - 037 in process

      Exchange in process

      Vj reserved

      14x in process, unit busy (VL) + 4 CPs

      003 in process, unit busy 3 CPs

      175 in process, unit busy (VL) + 4 CPs

Execution time

      Instruction issue 1 CP

      Vj ready 5 CPs if (VL) $\leq$ 5

      Vj ready (VL) CPs if (VL) > 5

      Unit ready except for 073 instruction (VL) + 4 CPs

      Unit ready for 073 instruction (VL) + 6 CPs

Special cases

      k = 0 or 4, VM bit xx = 1 if (Vj element xx) = 0

      k = 1 or 5, VM bit xx = 1 if (Vj element xx) $\neq$ 0

      k = 2 or 6, VM bit xx = 1 if (Vj element xx) is positive

      k = 3 or 7, VM bit xx = 1 if (Vj element xx) is negative

```
+---------------------------------------------------------------+
|                                                               |
|  176ixk    Transmit (VL) words from memory to Vi elements     |
|            starting at memory address (A0) and incrementing   |
|            by (Ak) for successive addresses                   |
|  177xjk    Transmit (VL) words from Vj elements to memory     |
|            starting at memory address (A0) and incrementing   |
|            by (Ak) for successive addresses                   |
|                                                               |
+---------------------------------------------------------------+
```

These instructions transfer blocks of data between V registers and memory.
The 176 instruction transfers data from memory to elements of register Vi.
The 177 instruction transfers data from elements of register Vj to memory.
Register elements begin with zero and are incremented by one for each
transfer. Memory addresses begin with (A_0) and are incremented by the
contents of Ak. Ak contains a signed integer which is added to the
address of the current word to obtain the address of the next word. Ak
may specify either a positive or negative increment allowing both forward
and backward streams of reference.

The number of words transferred is determined by the contents of the VL
register.

Hold issue conditions

     034 - 037 in process

     Exchange in process

     A_0 reserved

     Ak reserved where k = 1 through 7

     Block sequence flag set (034 - 037, 176, 177)

     Scalar reference

     Rank B data valid

     Fetch request in last clock period

     For 176, vector register i reserved

     For 177, vector register j reserved

     I/O memory request

Execution time

For 176:

Instruction issue except for 034-037, 100-137, 176, 177:   1 CP

Instruction issue for above exceptions: (VL) + 4 CPs

Vi ready 14 CPs if (VL) $\leq$ 5

Vi ready (VL) + 9 CPs if (VL) > 5

For 177:

Instruction issue except for 034-037, 100-137, 176, 177:   1 CP

Instruction issue for above exceptions: (VL) + 5 CPs

Vj ready 5 CPs if (VL) $\leq$ 5

Vj ready (VL) CPs if (VL) > 5

Special cases

The increment, (A_0), = 1 if k = 0

Chain slot issue is 9 CPs if full speed for 176, blocked for 177

Block I/O references

Block 034 - 037, 100 - 137, 176, 177

(Ak) determines speed control. There are 16 memory banks;
successive addresses are located in successive banks. References
to the same bank can be made every 4 CPs or more. Incrementing
(Ak) by 16[†] places successive memory references in the same bank,
so a word is transferred every 4 CPs. If (Ak) is incremented
by 8,[††] every other reference is to the same bank and words can
transfer every 2 CPs. With any address incrementing that allows
4 CPs before addressing the same bank, the words can transfer
each CP.

Memory reference out of limits will allow 6 CPs + 2 parcels to issue.

For 176, a parity error will allow a minimum of 16 CPs + 2 parcels
to issue and a maximum of (VL) + 15 CPs + 2 parcels to issue.

[†] 8 places for 8-bank memory option. Refer to section 5.

[††] 4 places for 8-bank memory option. Refer to section 5.

SECTION 5

MEMORY SECTION

INTRODUCTION

The memory for the CRAY-1 normally consists of 16 banks[†] of bi-polar LSI memory. Three memory sizes are available:

262,144 words,
524,288 words, or
1,048,576 words.

The banks are independent of each other.

MEMORY CYCLE TIME

The memory cycle time is four clock periods (50 nsec). The access time, that is, the time required to fetch an operand from memory to an operational register is 11 clock periods (127.5 nsec).

MEMORY ACCESS

The memory of the CRAY-1 Computer System is shared by the computation section and the I/O section. A single port access is provided.

Because of the interleaving scheme used to address the independent banks, it is possible to reference memory every clock period with a new request. It is not possible, however, to reference any one bank sooner than its 4 CP cycle time. Trying to reference a bank more often than every 4 CPs causes memory conflicts. These conflicts are handled in an orderly, predictable manner.

All block transfers require memory to be quiet before issuing. Once issued, they block all other memory requests. Multiple block transfers cannot issue without allowing one waiting I/O reference to complete. The maximum duration of a lockout caused by block transfers is one block length.

Vector block transfers may conflict with themselves. Therefore, the vector logic provides for identifying these conditions (speed control) and for

[†] See eight-bank phasing.

slowing or disallowing the vector operations that would be affected by the slowed memory referencing rate. The vector logic identifies 1/4 speed (4 CP), 1/2 speed (2 CP), and full speed (1CP) data rates from memory.

Fetch operations require memory to be quiet before referencing memory. Once the fetch request is honored, all other memory references are blocked.

Exchange operations require memory to be quiet before referencing memory. After the exchange has issued, all other memory references are blocked.

Scalar and I/O memory references are examined in three registers for possible memory conflicts. These three registers contain the lower 4 bits[†] of each of the referenced memory addresses. These registers plus the address register represent the 4 CPs between referencing any one bank. The first bank is rank A, the second is rank B, and the third is rank C. At each clock, the contents of the registers are shifted down one rank until they are discarded unless a conflict arises, in which case the conflicting address is held in rank B until the conflict is resolved.

I/O requests are tested against ranks A, B, and C. Coincidence with rank A, B, or C disallows the request. An I/O request that is disallowed must wait eight clock periods before it can request again.

The following conditions must be present for an I/O memory request to be processed:

1. I/O request
2. No coincidence in rank A, B, or C
3. No scalar memory reference instruction in clock period two of its sequence (scalar priority over I/O)
4. No fetch request
5. No 176, 177, or 034 through 037 instruction in progress
6. No exchange sequence
7. No 033 request (not a memory conflict)

Scalar instruction memory requests are tested in ranks A, B, and C for memory conflicts. Scalar instructions have priority over I/O requests arriving at memory in the same clock period.

[†] See eight-bank phasing.

A scalar conflict in rank A (CP 2 of a scalar instruction) causes a hold storage on this instruction for three clock periods. At the same time, a hold issue signal blocks the issue of another scalar reference instruction. The only memory conflict that may occur in rank A is a scalar reference conflicting with a previous I/O reference. It is not possible for a scalar to conflict with a scalar in rank A because it takes two clock periods to issue a scalar reference instruction.

A scalar conflict in rank B (CP 3) causes a hold storage on this instruction for two clock periods. Also, a hold issue signal blocks issue of another scalar reference instruction.

A scalar conflict in rank C (CP 4) causes a hold storage on this instruction for one clock period. There is also a hold issue signal, which blocks issue of another scalar reference instruction.

Under normal operating conditions on codes performing a mix of vector and scalar instructions, the memory access will support four disk and three interface channels without degrading the CPU computation rate. However, a single program requiring memory access continuously will be measurably degraded by maximum I/O transfer conditions. This is caused by the delays imposed on the issue of vector memory instructions because block transfers require memory quiet before issue.

MEMORY ORGANIZATION

The memory is organized into 8 or 16 interleaved banks to minimize memory conflicts and to exploit the speed of the memory chip. Each bank occupies a chassis and contains 72 modules. Each module contributes one data or check bit to each 72-bit word in the bank; a memory word consists of 64 data bits and 8 check bits.

The 16-bank phasing is standard on the CRAY-1; 8-bank phasing, allowing a maximum memory size of 1/2 million owrds, can be accomplished by replacing two modules and setting the bank select switch to the left or the right banks. This option is available on any 16-bank memory machine.

MEMORY ADDRESSING

A word in a 16-bank memory is addressed in 20 bits as shown in figure 5-1. The low order four bits specify one of the 16 banks.
The next field specifies an address within the chip.
The upper bits specify one of the chips on the module.

```
 2^19                                      2^3    2^0
┌─────────┬──────────────────────┬──────────┐
│ chip    │    bit address       │  4-bit   │
│ address │      in chip         │  bank    │
└─────────┴──────────────────────┴──────────┘
```

Figure 5-1. Memory address; 16 banks

A word in a 1/2 million word 8-bank memory is addressed in 19 bits (not shown):

The low order three bits specify one of the 8 banks
The next field specifies an address within the chip
The upper bits specify one of the chips on the module.

Addressing a full million words with 8-bank phasing is possible. In this case, the right/left bank select switch determines only whether the lower half of memory or the upper half is selected first in the addressing scheme by inverting or not inverting bit 2^{19}. Under program control, bit 2^{19} selects the lower or upper half of memory because the bit is injected at bit 2^1 of the memory address.

SPEED CONTROL

For 176 and 177 instructions, (Ak) determines speed control (table 5-1).

Table 5-1. Vector memory rate * 80 x 10^6 references per second

Phasing	Increment or multiple							
	1-3	4	5-7	8	9-11	12	13-15	16
8-bank	1	1/2	1	1/4	1	1/2	1	1/4
16-bank	1	1	1	1/2	1	1	1	1/4

For eight banks, incrementing 8 places causes successive references in the same bank so that a word is transferred every 4 CPs. If (Ak) is incremented by 4, an 8-bank memory transfers words every 2 CPs.

8-BANK PHASING OPTION

The 8-bank phasing option makes possible a system consisting of one-half million words arranged in only eight banks. Any 16-bank system can exercise the option by replacing two modules and setting the bank select switch to the left or right banks. A system constructed with only eight banks of modules but with all 12 of its columns can be upgraded to a 16-bank full million words by completing the remaining banks.

The effect of 8-bank phasing on instruction fetches is a predictable increase of 4 clock periods for filling an instruction buffer. Otherwise, the amount of performance degradation for 8 banks as compared with 16 banks is not readily predictable since it largely results from an increase of memory conflicts for vector memory references.

For other differences, refer to the preceding paragraphs on MEMORY ADDRESSING and SPEED CONTROL.

MEMORY PARITY ERROR CORRECTION

An error correction and detection network between the CPU and memory assures that the data written into memory can be returned to the CPU with consistent precision. (Refer to figure 5-2.)

The network operates on the basis of single error correction, double error detection (SECDED). If one bit of a data word is altered, the single error alteration is automatically corrected before passing the data word to the computer. If two bits of the same data word are altered, the double error is detected but not corrected. In either case, the CPU may be interrupted depending on interrupt options selected to prevent incorrect data from contaminating a job. For three or more bits in error, results are ambiguous.
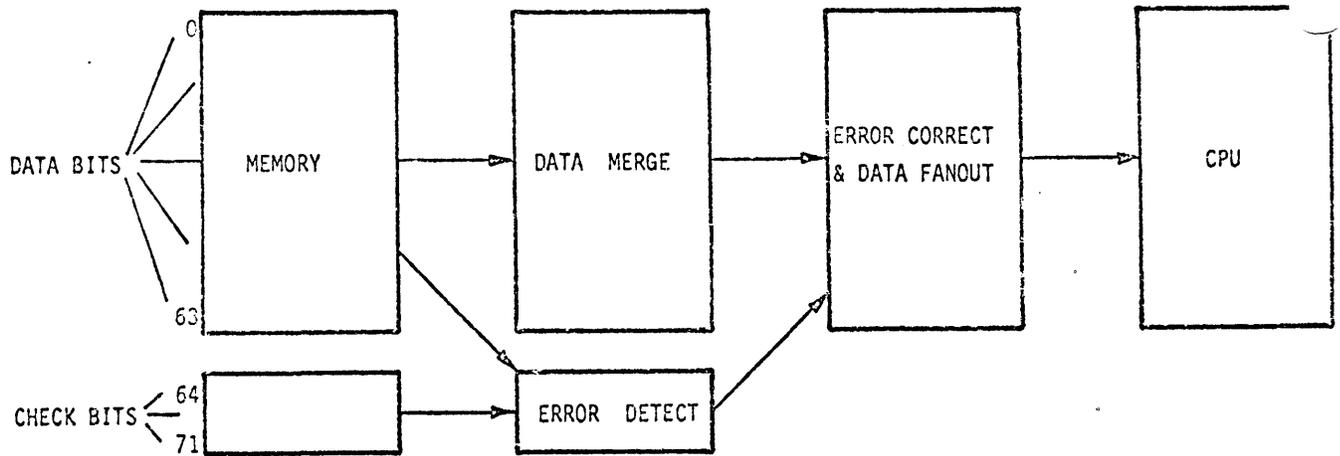
Figure 5-2. Memory data path with SECDED

The SECDED error processing scheme is based on error detection and correction codes devised by R. W. Hamming[†]. An 8-bit check byte is appended to the 64-bit data word before the data is written in memory. The eight check bits are each generated as even parity bits for a specific group of data bits. Figure 5-3 shows the bits of the data word used to determine the state of each check bit. An X in the horizontal row indicates that data bit contributes to the generation of that check bit. Thus, check bit number 0 (bit 2^{64}) is the bit making group parity even for the group of bits 2^1, 2^3, 2^5, 2^7, 2^9, 2^{11}, 2^{13}, 2^{15}, 2^{17}, 2^{19}, 2^{21}, 2^{23}, 2^{25}, 2^{27}, 2^{29}, and 2^{31} through 2^{55}.

The eight check bits are stored in memory at the same location as the data word. When read from memory, the same 72-bit matrix of figure 5-3 is used to generate a new set of parity bits, which are even parity bits of the data word and the old check bits. The resulting eight parity bits are called syndrome bits, shown as bits 64 through 71 in figure 5-3.

[†] Hamming, R.W., "Error Detection and Correcting Codes". Bell System Technical Journal, 29, No., 2, 147-160 (April, 1950).

Figure 5-3. Error correction matrix

```
        BYTE 0                    BYTE 1                    BYTE 2                    BYTE 3
 0  1  2  3  4  5  6  7     8  9 10 11 12 13 14 15    16 17 18 19 20 21 22 23    24 25 26 27 28 29 30 31
    x     x     x     x        x     x     x     x        x     x     x     x        x     x     x     x
       x  x        x  x           x  x        x  x           x  x        x  x           x  x        x  x
             x  x  x  x                 x  x  x  x                 x  x  x  x                 x  x  x  x
    x  x     x        x        x  x     x        x        x  x     x        x        x  x     x        x
 x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x
 x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x                                   x  x  x  x  x  x  x  x
 x  x  x  x  x  x  x  x                                x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x
                           x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x


        BYTE 4                    BYTE 5                    BYTE 6                    BYTE 7
32 33 34 35 36 37 38 39    40 41 42 43 44 45 46 47    48 49 50 51 52 53 54 55    56 57 58 59 60 61 62 63
 x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x
 x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x                                x  x  x  x  x  x  x  x
 x  x  x  x  x  x  x  x                                x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x
                           x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x     x  x  x  x  x  x  x  x
    x     x     x     x     x     x     x     x        x     x     x     x        x     x     x     x
       x  x        x  x        x  x        x  x           x  x        x  x           x  x        x  x
             x  x  x  x                 x  x  x  x                 x  x  x  x                 x  x  x  x
    x  x     x        x     x  x     x        x        x  x     x        x        x  x     x        x


                                                                       CHECK BYTE
                                                                  64 65 66 67 68 69 70 71
                                                                   x                        S0
                                                                      x                     S1
                                                                         x                  S2
                                                                            x               S3
                                                                               x            S4
                                                                                  x         S5
                                                                                     x      S6
                                                                                        x   S7
```

Figure 5-3. Error correction matrix

The states of these "S" bits are all symptoms of any error that occurred.

The matrix is designed so that any change of state of one data bit will change an odd number of syndrome bits. An error in two columns changes the parity states of an even number of bit groups. Therefore, a double error appears as an even number of syndrome bits set to 1.

The matrix is designed so that SECDED decodes the syndrome bits and determines the error condition using the following

    1. If all syndrome bits are 0, no error occurred.

    2. If only one syndrome bit is 1, the associated check bit is in error.

3. If more than one syndrome bit is 1 and the parity of
    all syndrome bits S0 through S7 is even, then a double
    error occurred within the data bits or check bits.

4. If more than one syndrome bit is 1 and the parity of all
    syndrome bits is odd, then a single and correctable error
    is assumed to have occurred. The syndrome bits can be
    decoded to identify the bit in error.

5. Results are ambiguous for three or more bits in error.

SECTION 6

INPUT/OUTPUT SECTION

<u>I/O CHANNELS</u>

The Input/Output section of the CRAY-1 contains 24 I/O channels of which twelve are input channels and twelve are output channels. The channels are assigned the numbers 2 through 31_8.

Three basic types of control logic for I/O channels are available:

    1. 16-bit asynchronous, for which three versions exist and are identified by their module types, as follows:

        a. DJ/DK module, used for MCU interface only

        b. DU/DK module, used for interfacing other devices (normal)

        c. DV/DK module, used for interfacing other devices (special)

    2. 16-bit high-speed asynchronous

    3. 16-bit synchronous (disk channel)

Each type of channel has the same electrical interface to the I/O cable but differs in timing, protocol, and data rates.

CHANNEL GROUPS

Channels are divided into four groups, as follows:

Group 1	Input channels	2, 6, 12, 16, 22, 26
Group 2	Output channels	3, 7, 13, 17, 23, 27
Group 3	Input channels	4, 10, 14, 20, 24, 30
Group 4	Output channels	5, 11, 15, 21, 25, 31

I/O INSTRUCTIONS

The instructions used with I/O channels are:

0010jk	Set the current address (CA) register for the channel indicated by (Aj) to (Ak) and activate the channel
0011jk	Set the limit address (CL) register for the channel indicated by (Aj) to (Ak)
0012jx	Clear the interrupt flag and error flag for the channel indicated by (Aj)
0033ijk	Transmit I/O status to Ai

BASIC CHANNEL OPERATION

Each input or each output channel directly accesses the CRAY-1 memory.
Input channels store external data in memory and output channels read
data from memory. A primary task of a channel is to convert 64-bit memory
words into 16-bit parcels or 16-bit parcels into 64-bit memory words. Four
parcels make up one memory word, with bits of the parcels assigned to
memory bit positions as shown in table 6-1. In both input and output
operations, parcel 0 is always transferred first.

Each channel consists of a data channel (4 parity bits, 16 data bits, and
3 control lines), a 64-bit assembly or disassembly register, a current
address register, and a limit address register.

The three control signals are: ready, resume, and disconnect. These
control signals coordinate the transfer of parcels over the channels.
The method of coordination varies among the types of channel; the dif-
ferent methods are explained later.

In addition to the three control signals, some channels have a master clear
line. The DJ, DU, and DV module input channels (asynchronous) have master
clear lines. The DO module output channel (high-speed asynchronous) has a
master clear line. The SI module output channel (synchronous) has a mas-
ter clear line.

Table 6-1. Channel word assembly/disassembly

Characteristic	Bit position	Number of bits	Comment
Channel data bits	$2^{15} - 2^0$	16	Four 4-bit groups
Channel parity bits		4	One per 4-bit group
CRAY-1 word	$2^{63} - 2^0$	64	
Parcel 0	$2^{63} - 2^{48}$	16	First in or out
Parcel 1	$2^{47} - 2^{32}$	16	Second in or out
Parcel 2	$2^{31} - 2^{16}$	16	Third in or out
Parcel 3	$2^{15} - 2^0$	16	Fourth in or out

I/O interrupts can be caused by the following:

- On all output channels, if (CA) becomes equal to (CL), then for each of the module types on the transmission of the last four parcels:

     DK module - Resume for last parcel sets interrupt
     DO module - Resume for last word sets interrupt
     SI module - Interrupt sets when last Ready is sent.

- (CA) becomes equal to (CL) on DV input module.

- External device disconnect received on any input channel.

- Channel error condition (described later in this section).

The number of the channel causing an interrupt can be determined by the use of a 033 instruction which reads to Ai the highest priority channel number requesting an interrupt. The lowest numbered channel has the highest priority. The interrupt request continues until cleared by the monitor program at which time an interrupt from the next highest priority channel, if present, may be sensed.

INPUT CHANNEL PROGRAMMING

To start an input operation, the CRAY-1 program must perform the following steps:

1. Set the channel limit address to the last word address+1 (LWA+1). See figure 6-1.
2. Set the channel current address to the first word address (FWA).

Setting the current address causes the channel active flag to be set and the channel is then ready to receive data. When a 4-parcel word is assembled, the word is stored in memory at the address contained in the channel current address register. When the word is accepted by memory, the current address is advanced by 1.

The external transmitting device sends a disconnect pulse to indicate the end of the transfer. When the disconnect is received, the channel interrupt flag sets and a test is performed to check for a partially
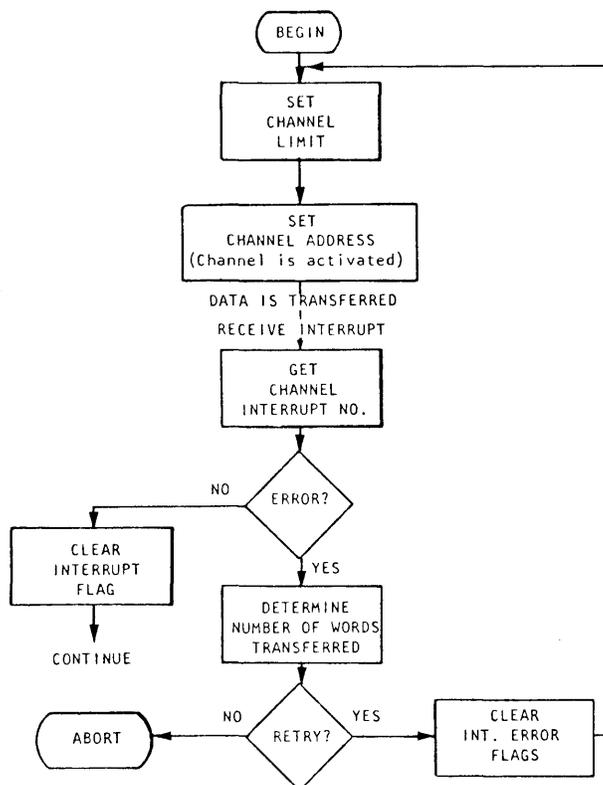
Figure 6-1. Basic I/O program flow chart

assembled word. If a partial word is found, the valid portion of the word
is stored in memory and the unreceived, lower-order parcels are stored as
zeros.   For the DV module, (CA) = (CL) causes the I/O interrupt request
unless the disconnect is received before the word count is exhausted.

The interrupt flag sets when a disconnect pulse is received or when an
error condition is detected. Setting the interrupt flag deactivates the
input channel.

Input channel error conditions

    1. Parity error

        DJ/DK asynchronous channel (MCU channel) - The parcel in which
        the error occurs will immediately set the channel error flag,
        deactivate the channel and generate an I/O interrupt request.
        If the error occurred in parcel 0, 1, or 2, the last 64-bit
        word is not stored. All input ready pulses received after
        the channel is deactivated are resumed but the data parcels
        are discarded.

SH/SI synchronous channel (disk channel) - The parcel in which the error occurs causes a parity fault flag to set. When parcel 3 arrives, or if parcel 3 is in error, a memory reference is initiated and the parity fault flag causes the channel error flag to set which in turn generates an I/O interrupt request. The channel error flag also deactivates the channel. Data parcels received after the parcel in error are not sampled. Parcels received up to and including the parcel in error are stored in memory. Any un-sampled lower-order parcels are stored as zeros. Once the channel is deactivated, no more resume pulses are sent to the DCU to request the remainder of the data block.

All other channels - The channel samples and stores the data until the parcel containing the error is received. At this time, the channel error flag is set and the data transfer proceeds as if no error had occurred. The transfer continues until the disconnect occurs or until (CA) = (CL) for a DV module channel. The interrupt is then generated and the channel is deactivated.

2. Unexpected ready pulse

   DV/DK asynchronous channel - Data is held and the resume occurs when the channel is reactivated. No error interrupt is generated.

   SH/SI synchronous channel (disk channel) - The data is resumed and thrown away. An error interrupt is generated. This channel uses this method to flag fire code errors.

   All other channels - The data is resumed and thrown away. An error interrupt is generated.

DU Module

The input channel control logic for the DU module differs from the DJ module in two respects.

1. When a parity error is detected, the condition is noted and saved but the Channel Error Flag (CE) is not set until the Input Disconnect pulse arrives. This change prevents an error interrupt request from occurring and no data is lost. The only interrupt request that occurs in this situation is the normal one at disconnect time, even though the Channel Error Flag is set at this time to indicate the parity fault condition.

2. For the DU module, the input channel is not forced active by the clear I/O signal. If, however, the channel is already active, it remains active.

DV module

The input channel control logic for the DV module differs from that for the DJ module in six respects.

1. When a parity error is detected, the condition is noted and saved but the Channel Error Flag (CE) is not set until the Input Disconnect pulse arrives. This change prevents an error interrupt request from occurring and no data is lost. The only interrupt request that occurs in this situation is the normal one at disconnect time, even though the Channel Error Flag is set at this time to indicate the parity fault condition.

2. For the DV module, the input channel is not forced active by the Clear I/O signal. If, however, the channel is already active, it remains active.

3. In an Input Ready pulse is received while the input channel is not active, even if (CA) = (CL), the ready is held until the channel goes active or until a Master Clear is received, (i.e., a Clear I/O signal is generated by the MCU or a Programmed I/O Master Clear sequence is performed). No error interrupt request is made.

4. If the channel address (CA) equals the limit address (CL) and the input channel is active, an interrupt request is generated and the input channel goes inactive without receiving an Input Disconnect pulse. When the Disconnect pulse is received after (CA) = (CL), it is ignored since the interrupt request has already been generated.

5. The only conditions that cause the Channel Error (CE) flag to set are:

    a. Input Ready and Reference; double Ready condition

    b. Input Ready and Active and (CA) = (CL); double Ready condition

    c. Parity Fault Flag set and Disconnect

    d. Parity Fault Flag set and Active and (CA) = (CL)

6. The Clear I/O signal clears the Parity Fault flag.

OUTPUT CHANNEL PROGRAMMING

To start an output operation, the CRAY-1 program must:

1. Set the channel limit address to the last word address + 1 (LWA+1)
2. Set the channel current address to the first word address (FWA).

Setting the current address causes the channel active flag to be set. The channel reads the first word from memory addressed by the contents of the channel's current address register. When the word is received from memory, the channel advances the current address by one and starts the data transfer.

After each word is read from memory and the current address is advanced, a limit test is made. The test compares the contents of the channel's current address register and the channel's limit address register. If they are equal, the transfer is completed as soon as the present word is transferred. Then, a disconnect pulse is sent to indicate the end of the transfer.

When the disconnect pulse is sent, the channel is deactivated and an I/O interrupt request is generated by the channel.

Output channel error condition

The interrupt flag also sets if an error is detected. The only error that an output channel detects is a resume pulse received when the channel is not active.

16-BIT ASYNCHRONOUS CHANNELS

Input channels

Table 6-2 illustrates a general view of an input signal sequence.

> Data Bits 2^0 through 2^{15} - Data Bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the external device to the CRAY-1. They must all be valid within 80 nanoseconds after the leading edge of the Ready signal. Data Bit signals must remain unchanged on the lines until the corresponding resume is received by the external device. Normally, data is sent coincident with the Ready pulse and is held until the subsequent Ready pulse.

Table 6-2 16-bit asynchronous input channel signal exchange
(DJ, DU, or DV modules)

CRAY-1	External
1. Activate channel (Set CL and CA).	
2.	◄——— Data 2^{63}-2^{48} with Ready
3. Resume ——►	
4.	◄——— Data 2^{47}-2^{32} with Ready
5. Resume ——►	
6.	◄——— Data 2^{31}-2^{16} with Ready
7. Resume ——►	
8.	◄——— Data 2^{15}-2^0 with Ready
9. Write word to memory and advance current address.	
10a. Resume ——►	
10b. For DV only, if (CA) = (CL), go to 13.	
11.	If more data, go to 2.
12.	◄——— Disconnect
13. Set interrupt and deactivate channel.	

Parity Bits 0 through 3 - Parity Bits 0 through 3 are each assigned
to a 4-bit group of data bits. The parity bits are set or cleared to
give the bit group odd parity. Bit assignments are as follows:

> Parity Bit 0   Data Bits 2^0 - 2^3
>
> Parity Bit 1   Data Bits 2^4 - 2^7
>
> Parity Bit 2   Data Bits 2^8 - 2^{11}
>
> Parity Bit 3   Data Bits 2^{12} - 2^{15}

Parity bits are sent from the external device to the CRAY-1 at the
same time as the data bits. They are held stable in the same way as
are the data bits.

Ready - The Ready signal sent to the CRAY-1 indicates that a parcel
of data is being sent to the CRAY-1 input channel and may be sampled.
The Ready signal is a pulse 50 + 10 nanoseconds wide (at 50% voltage
points). The leading edge of Ready at the CRAY-1 begins the timing
for sampling the data bits.

Resume - Resume is sent from the CRAY-1 to the external device to show
that the parcel was received and that the CRAY-1 is ready for the next
data transmission. Resume is a pulse 50 ± 3 nanoseconds wide (at 50%
voltage points).

Disconnect - This signal is sent from the external device to the CRAY-1 and means that the transmission from the external device is complete. It is sent after the Resume is received for the last Ready. Disconnect is a pulse 50 $\pm$ 10 nanoseconds wide (at the 50% voltage points).

Channel Master Clear - This signal may be programmed (see description of Programmed Master Clear later in this section) or may result from a Clear I/O Signal.

Output channels

Table 6-3 illustrates a general view of an output signal sequence.

Table 6-3.   16-bit asynchronous output channel signal exchange
(DK module)

CRAY-1	External
1. Activate channel (set CL and CA)	
2. Read word from memory and advance current address	
3. Data 2^{63}-2^{48} with Ready ⟶	
4.	⟵ Resume
5. Data 2^{47}-2^{32} with Ready ⟶	
6.	⟵ Resume
7. Data 2^{31}-2^{16} with Ready ⟶	
8.	⟵ Resume
9. Data 2^{15}-2^0 with Ready ⟶	
10.	⟵ Resume
11. If (CA) $\neq$ (CL), go to 2.	
12. Disconnect ⟶	
13. Set interrupt and deactivate channel.	

Data Bits 2^0 through 2^{15} - Data Bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the CRAY-1 to an external device. They are all sent at the same time, within 5 nanoseconds of the leading edge of the Ready pulse. Data Bit signals remain steady on the lines until the next parcel is sent.

Parity Bits 0 through 3 - Parity Bits 0, 1, 2, and 3 are each assigned to a 4-bit group of data bits. The parity bits are set or cleared to give the bit broup odd parity. Bit assignments are as follows:

Parity Bit 0   Data Bits 2^0 - 2^3

Parity Bit 1   Data Bits 2^4 - 2^7

Parity Bit 2   Data Bits 2^8 - 2^{11}

Parity Bit 3   Data Bits 2^{12} - 2^{15}

Parity bits are sent from the CRAY-1 to the external device at the same time as the data bits. They are held stable in the same way as are the data bits.

Ready - The Ready signal sent from the CRAY-1 to the external device indicates that the data is present and may be sampled. The Ready signal is a pulse 50 ± 3 nanoseconds wide (at 50% voltage points). The leading edge of Ready may be used to time data sampling in the external device.

Resume - Resume is sent from the external device to the CRAY-1 to show that the parcel was received and that the external device is ready for the next parcel transmission. Resume is a pulse 50 ± 10 nanoseconds wide (at 50% voltage points).

Disconnect - Disconnect is a signal sent from the CRAY-1 to the external device that means the transmission from the CRAY-1 is complete. It is sent after the CRAY-1 has received the Resume from the last Ready. The Disconnect is a pulse 50 ± 3 nanoseconds wide (at 50% voltage points).

16-BIT HIGH-SPEED ASYNCHRONOUS CHANNELS

Input channels

Table 6-4 illustrates a general view of an input signal sequence.

Data Bits 2^0 through 2^{15} - Data Bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the external device to the CRAY-1. The data lines must be stable no later than 80 nanoseconds after the leading edge of the associated Ready pulse and must be held stable until at least 120 nanoseconds after the leading edge of the same Ready. Note that if the device is transmitting at the maximum allowable rate, it is normal for a data parcel to overlap the subsequent Ready pulse. Typically, data is transmitted 50 nsec after the leading edge of Ready and held until 50 nsec after the leading edge of the following Ready pulse.

Parity Bits 0 through 3 - Parity Bits 0, 1, 2, and 3 are each a parity bit assigned to a 4-bit group of data bits. The parity bits are set or cleared to give the bit group odd parity.

Table 6-4. 16-bit high-speed asynchronous input channel signal exchange
(DN module)

CRAY-1	External
1. Activate channel (set CL and CA)	
2. Resume ——▶	
3. Resume ——▶	
4. Resume ——▶	
5. Resume ——▶	If done, go to 11.
6.	◀—— Data 2^{63}- 2^{48} with Ready
7.	◀—— Data 2^{47}- 2^{32} with Ready
8.	◀—— Data 2^{31}- 2^{16} with Ready
9.	◀—— Data 2^{15}- 2^{0} with Ready
10. Write word to memory and advance current address; go to 2.	
11.	◀—— Disconnect
12. Set interrupt and deactivate channel.	

Bit assignments are as follows:

    Parity Bit 0   Data Bits 2^0 - 2^3

    Parity Bit 1   Data Bits 2^4 - 2^7

    Parity Bit 2   Data Bits 2^8 - 2^{11}

    Parity Bit 3   Data Bits 2^{12} - 2^{15}

Parity bits are sent from the external device to the CRAY-1 at the same time as the data bits. They are held stable in the same way as are the data bits.

Ready - The Ready signal sent to the CRAY-1 indicates that data will soon be sent to the CRAY-1 input channel and may be sampled. The Ready signal is a pulse 50 ± 10 nanoseconds wide (at the 50% voltage points) sent in groups of four. The leading edge of Ready at the CRAY-1 begins the timing for sampling the data bits.

The time from the leading edge of one Ready pulse to the leading edge of the following Ready pulse in the same group must be greater than 90 nsec. The first Ready pulse of a group may be transmitted by the device as soon as it detects the leading edge of the first Resume pulse for that group.

Resume - This signal is sent to the external device to show that the CRAY-1 is ready for the next data transmission. Resume is a pulse 50 ± 3 nanoseconds wide (at the 50% voltage points) sent in groups of four.

For any group of Resume pulses, the time from the leading edge of one Resume to the leading edge of the next Resume is 100 ± 3 nsec.

Disconnect - This signal is sent from the external device to the CRAY-1 and indicates that the transmission from the external device is complete. It is sent after the last Ready. The Input Disconnect pulse must be transmitted no earlier than 20 nsec after the leading edge of the final Ready pulse. Disconnect is a pulse 50 ± 10 nanoseconds wide (at the 50% voltage points).

Output channels

Table 6-5 illustrates a general view of an output signal sequence.

Table 6-5. 16-bit high-speed asynchronous output channel signal exchange
(DO module)

CRAY-1	External
1. Activate channel (set CL and CA).	
2. Read word from memory and advance current address.	
3. Data 2^{63}-2^{48} with Ready $\longrightarrow$	
4. Data 2^{47}-2^{32} with Ready $\longrightarrow$	
5. Data 2^{31}-2^{16} with Ready $\longrightarrow$	
6. Data 2^{15}-2^{0} with Ready $\longrightarrow$	
(with Disconnect if this is the last word)	
7.	$\longleftarrow$ Resume
8. If (CA) $\neq$ (CL), go to 2.	
9. Set interrupt and deactivate channel.	

Data Bits 2^0 through 2^{15} - Data Bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the CRAY-1 to an external device. They are all sent at the same time, within 5 nanoseconds of the leading edge of the Ready pulse. Data Bit signals remain steady on the lines until the next parcel is sent.

Parity Bits 0 through 3 - Parity Bits 0, 1, 2, and 3 are each assigned to a 4-bit group of data bits. The parity bits are set or cleared to give the bit group odd parity. Bit assignments are as follows:

Parity Bit 0	Data Bits 2^0 - 2^3
Parity Bit 1	Data Bits 2^4 - 2^7
Parity Bit 2	Data Bits 2^8 - 2^{11}
Parity Bit 3	Data Bits 2^{12} - 2^{15}

Parity bits are sent from the CRAY-1 to the external device at the same time as the data bits. They are held stable in the same way as are the data bits.

Channel Master Clear - The Channel Master Clear may be programmed (see description of Programmed Master Clear later in this section) or may the result of a Clear I/O signal. The Master Clear signal may be used by the external devices for control purposes or may be ignored.

Ready - The Ready signal sent from the CRAY-1 to the external device indicates that the data is present and may be sampled. The Ready signal is a pulse 50 ± 3 nanoseconds wide (at the 50% voltage points) sent in groups of four. For any group of Ready pulses, the time from the leading edge of one Ready to the leading edge of the next Ready is 100 ± 3 nanoseconds. The leading edge of Ready may be used to time data sampling in the external device.

Resume - Resume is sent from the external device to the CRAY-1 to show that the 64-bit word of four parcels was received and that the external device is ready for the next word (four parcels). Resume is a pulse 50 ± 10 nanoseconds wide (at the 50% voltage points). The pulse must be received at the CRAY-1 no earlier than 230 nanoseconds after the leading edge of the first Ready pulse is transmitted.

Disconnect - Disconnect is a signal sent from the CRAY-1 to the external device that means the transmission from the CRAY-1 is complete. It is sent with the last Ready ± 3 nanoseconds. The Disconnect pulse is 50 ± 3 nanoseconds wide (at the 50% voltage points).

16-BIT SYNCHRONOUS CHANNELS

Input channels

Table 6-6 illustrates a general view of an input signal sequence.

Data Bits 2^0 through 2^{15} - Data Bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the external device to the CRAY-1. They are all valid within 5 nanoseconds of each other. Data Bit signals must remain unchanged on the lines until the next parcel is sent.

Table 6-6.   16-bit synchronous input channel signal exchange
(SH module)
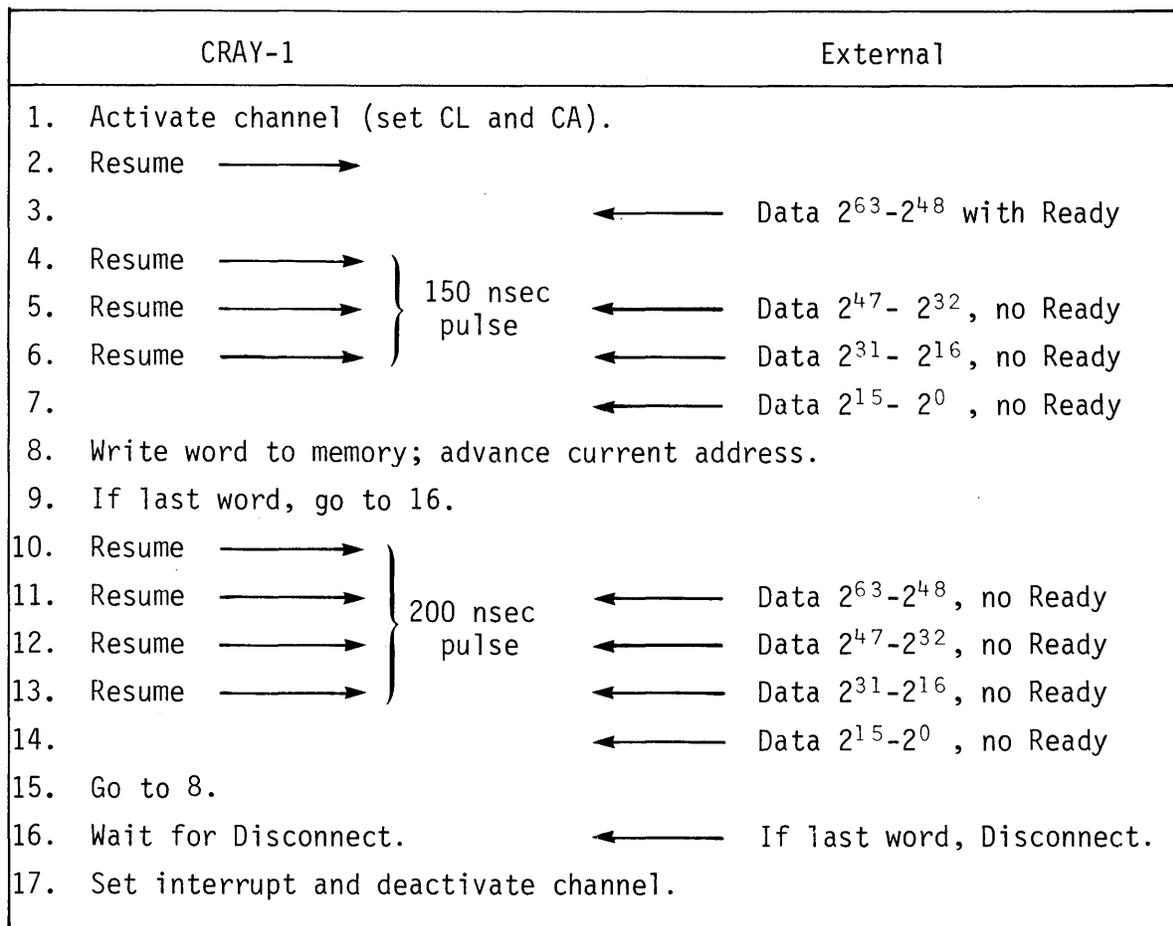
CRAY-1		External
1. Activate channel (set CL and CA).		
2. Resume ——————▶		
3.	◀——————	Data 2^{63}-2^{48} with Ready
4. Resume ——————▶	⎫	
5. Resume ——————▶	⎬ 150 nsec pulse	◀—————— Data 2^{47}- 2^{32}, no Ready
6. Resume ——————▶	⎭	◀—————— Data 2^{31}- 2^{16}, no Ready
7.	◀——————	Data 2^{15}- 2^{0} , no Ready
8. Write word to memory; advance current address.		
9. If last word, go to 16.		
10. Resume ——————▶	⎫	
11. Resume ——————▶	⎬ 200 nsec pulse	◀—————— Data 2^{63}-2^{48}, no Ready
12. Resume ——————▶		◀—————— Data 2^{47}-2^{32}, no Ready
13. Resume ——————▶	⎭	◀—————— Data 2^{31}-2^{16}, no Ready
14.	◀——————	Data 2^{15}-2^{0} , no Ready
15. Go to 8.		
16. Wait for Disconnect.	◀——————	If last word, Disconnect.
17. Set interrupt and deactivate channel.		

Parity Bits 0 through 3 - Parity Bits 0, 1, 2, and 3 are each assigned
to a 4-bit group of data bits. The parity bits are set or cleared to
give the bit group odd parity. Bit assignments are as follows:

Parity Bit 0	Data Bits 2^{0} - 2^{3}
Parity Bit 1	Data Bits 2^{4} - 2^{7}
Parity Bit 2	Data Bits 2^{8} - 2^{11}
Parity Bit 3	Data Bits 2^{12} - 2^{15}

Parity bits are sent from the external device to the CRAY-1 at the
same time as the data bits. They are held stable in the same way as
are the data bits.

Ready - The Ready signal is a block ready in response to the first resume of a block. The Ready signal is a pulse 50 ± 10 nanoseconds wide (at the 50% voltage points). It is sent from the external device to the CRAY-1.

Resume - Resume is sent from the CRAY-1 to the external device to initiate the synchronous data transfer and to time the sending of data at the CRAY-1. The Resume pulse is 50 ± 3 nanoseconds wide (at the 50% voltage points). Following the first resume, which awaits a ready response, the signal is sent in one group of three resumes followed by as many groups of four resumes as required to complete the block transfer.

Disconnect - Disconnect is a signal sent from the external device to the CRAY-1 indicating that transmission from the external device is complete. It is sent with parcel 2 of the last data word or at any later time. Disconnect is a pulse 50 ± 10 nanoseconds wide (at the 50% voltage points).

Block length restrictions - The input channel has no restrictions on block length. The mass storage controller, which is the only device connected to this type of channel, has rigid restrictions on its block lengths. Input transmissions are limited to 1 or 4 or 512 64-bit words.

Cabling restrictions - The synchronous channels use a fixed length cable providing constant propagation time for the signals. This cable delay is designed into the control logic; therefore, the cable length and propagation speed cannot be changed. The total cable length between the CRAY-1 and the external device is 17 feet (518 cm). The cable run for a synchronous channel uses one 10 foot (305 cm) drop cable at the CRAY-1 and one 7 foot (213 cm) length of data cable at the external device.

Clock - A clock signal is supplied over a separate cable (one per DCU cabinet) to the external device from the CRAY-1. This clock signal synchronizes signals at the external device interface connector.

Output channels

Table 6-7 illustrates a general view of an output signal sequence.

Data Bits 2^0 through 2^{15} - Data Bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the CRAY-1 to the external device. They are sent with the leading edge of the Ready pulse + 5 nsec. Data Bit signals remain unchanged on the lines until the next parcel is sent.

Parity Bits 0 through 3 - Parity Bits 0, 1, 2, and 3 are each assigned to a 4-bit group of data bits. The parity bits are set or cleared to

Table 6-7. 16-bit synchronous output channel signal exchange
(SI module)
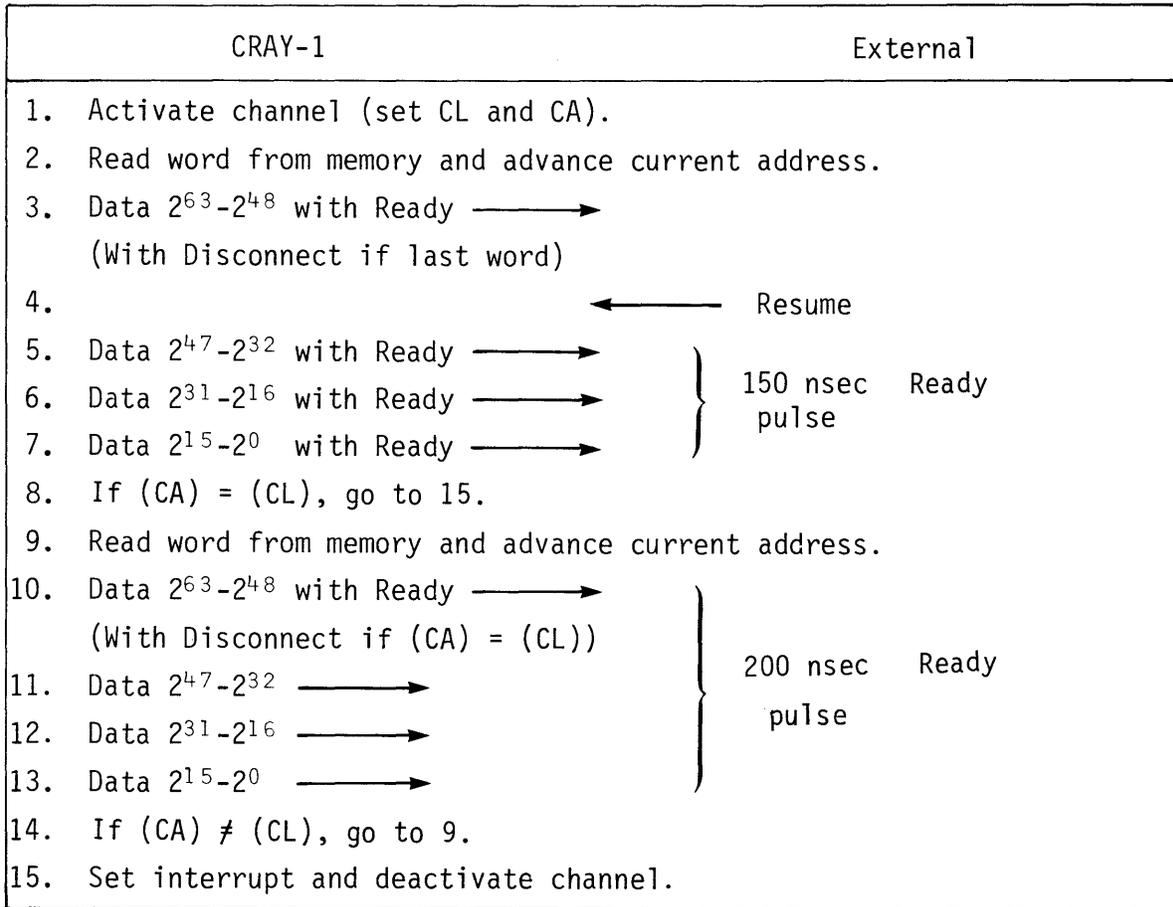
CRAY-1	External
1. Activate channel (set CL and CA).	
2. Read word from memory and advance current address.	
3. Data 2^{63}-2^{48} with Ready ⟶	
(With Disconnect if last word)	
4. ⟵ Resume	
5. Data 2^{47}-2^{32} with Ready ⟶	
6. Data 2^{31}-2^{16} with Ready ⟶ 150 nsec Ready pulse	
7. Data 2^{15}-2^{0} with Ready ⟶	
8. If (CA) = (CL), go to 15.	
9. Read word from memory and advance current address.	
10. Data 2^{63}-2^{48} with Ready ⟶	
(With Disconnect if (CA) = (CL))	
11. Data 2^{47}-2^{32} ⟶ 200 nsec Ready pulse	
12. Data 2^{31}-2^{16} ⟶	
13. Data 2^{15}-2^{0} ⟶	
14. If (CA) ≠ (CL), go to 9.	
15. Set interrupt and deactivate channel.	

give the bit group odd parity. Bit assignments are as follows:

Parity Bit 0	Data Bits 2^{0} - 2^{3}
Parity Bit 1	Data Bits 2^{4} - 2^{7}
Parity Bit 2	Data Bits 2^{8} - 2^{11}
Parity Bit 3	Data Bits 2^{12}- 2^{15}

Parity bits are sent from the CRAY-1 to the external device at the same time as the data bits. They are held stable in the same way as are the data bits.

Channel Master Clear - The Channel Master Clear may be programmed (see description of Programmed Master Clear later in this section) or may be the result of a Clear I/O signal. The programmed Master Clear to external is a static signal sent from the CRAY-1 to an external device. The Master Clear signal may be used by the external device for control purposes or it may be ignored.

Ready - The Ready signal is sent from the CRAY-1 to the external
device to indicate that the data is valid. The first Ready signal
is a pulse 50 ± 3 nanoseconds wide (at the 50% voltage points).
Following the first ready, which awaits a resume response, the signal
is sent in one group of three readies followed by as many groups
of four readies as required to complete the block transfer.

Resume - Resume is sent from the external device to the CRAY-1 in
response to the first Ready signal. The Resume pulse is 50 ± 10
nanoseconds wide (at the 50% voltage points).

Disconnect - Disconnect is a signal sent from the CRAY-1 to the
external device indicating that the transmission from the CRAY-1
is complete. It is sent with parcel 0 of the last 64-bit data word.
Disconnect is a pulse 50 ± 3 nanoseconds wide (at the 50% voltage
points).

Block length restrictions - The output channel has no restrictions
on block length. The mass storage controller, which is the only
device connected to this type of channel, has rigid restrictions on
its block lengths. Output transmissions are limited to 1 or 512
64-bit words.

Cabling restrictions - The synchronous channels use a fixed length
cable providing a constant propagation time for the signals. This
cable delay is designed into the control logic; therefore, the cable
length and propagation speed cannot be changed. The total cable length
between the CRAY-1 and the external device is 17 feet (518 cm). The
cable run for a synchronous channel uses one 10 foot (305 cm) drop
cable at the CRAY-1 and one 7 foot (213 cm) length of data cable at
the external device.

Clock - A clock signal is supplied over a separate cable (one per
DCU cabinet) to the external device from the CRAY-1. This clock
signal synchronizes signals at the external device interface connector.

PROGRAMMED MASTER CLEAR TO EXTERNAL

The CRAY-1 contains a mechanism for sending a Master Clear signal to an
external device.

Sequence for normal-speed channels

For the normal-speed asynchronous channels (DJ/DK, DU/DK, DV/DK), delays
1 and 2 are device dependent. For CRI interfaces, they sould be at least
1 microsecond.

External Master Clear sequence for 16-bit normal-speed asynchronous channel:

1. 0012 jk   Clear *output* channel to insure CRAY-1 activity on the
               channel pair has stopped.

2. 0012 jk   Clear *input* channel to insure external activity on the
               channel pair has stopped.

3. 0011 jk   Set the *input* channel limit to an arbitrary value.

4. 0010 jk   Set the *input* channel current address equal to the same
               value. This initiates the Master Clear signal.

5. 0012 jk   Clear the *input* channel. This stops the input channel
               activity just initiated.

6. Delay 1   Device dependent - this determines the duration of the
               Master Clear signal.

7. 0011 jk   Set the *input* channel limit. This value may be the same
               value as used in steps 3 and 4. This turns off the Master
               Clear signal.

8. Delay 2   Device dependent - this allows time for initialization
               activities in the attached device to complete.

Sequence for high-speed channels

For the high-speed synchronous channel (SH/SI), delay 1 should be a
minimum of 1 clock period and delay 2 a minimum of 20 clock periods.

External Master Clear sequence for high-speed synchronous and asynchronous
(DN/DO) channels:

1. 0012 jk   Clear *output* channel interrupt to assure that CRAY-1
               activity on the channel pair has stopped.

2. 0012 jk   Clear *input* channel interrupt to assure that external
               activity on the channel pair has stopped.

3. 0011 jk   Set the *output* channel limit to an arbitrary value.

4.   0010 jk   Set the *output* channel current address equal to the same value. This initiates the Master Clear signal.

5.   0012 jk   Clear the *output* channel. This stops the output channel activity just initiated.

6.   Delay 1   Device dependent - this determines the duration of the Master Clear signal.

7.   0011 jk   Set the *output* channel limit. This value may be the same value as used in steps 3 and 4. This turns off the Master Clear signal.

8.   Delay 2   Device dependent - this allows time for initialization activities in the attached device to complete.

9.   Read disk subsystem status (high-speed synchronous channel only). A subsystem status should be taken and discarded to remove any false status left by the Master Clear sequence.

MEMORY ACCESS

Each of the four channel groups is assigned a time slot (figure 6-2), which is scanned once every four clock periods for a memory request. The lowest-numbered channel in the group has the highest priority. A memory request, whether accepted or rejected, causes the requesting channel to miss the next time slot. Therefore, any given channel can request a memory reference only every eight clock periods. However, another channel in the same group as a channel that has just made a memory request can cause a memory request four clock periods later. During the next three clock periods, the scanner will allow requests from the other three channel groups. Therefore, it is possible to have an I/O memory request every clock period.
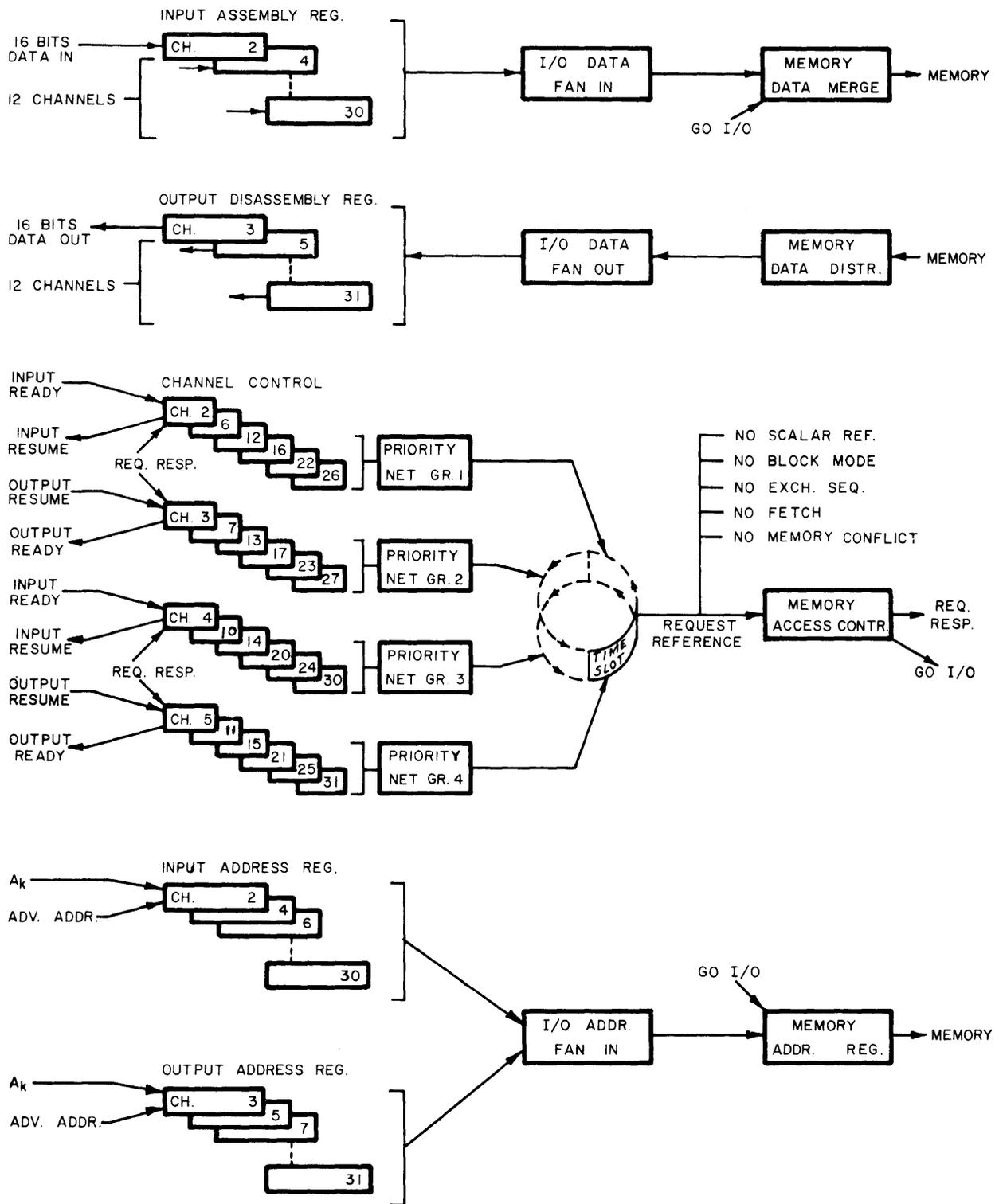
Figure 6-2. Channel I/O control

I/O LOCKOUT

An I/O memory request can be locked out by a block transfer. Multiple block transfers cannot issue without allowing one waiting I/O reference to complete. The maximum duration of a lockout caused by block transfers is one block length.

Exchange sequences and instruction fetch sequences can also cause lockouts.

MEMORY BANK CONFLICTS

Memory bank conflicts are tested for CPU scalar references and I/O memory references. All other memory references (block transfers, exchange sequences, instruction fetch sequences) wait issue until all memory banks are quiet. When a block transfer, exchange sequence, or instruction fetch sequence has issued, all other memory references are locked out.

Each memory bank can accept a new request every four clock periods. To test for a memory bank conflict, the lower four bits[†] of the memory address move through three 1-clock-period registers. The first register is rank A, the second is rank B, and the third is rank C. On the fourth clock, the address is placed in the memory address register.

I/O MEMORY CONFLICTS

Before coincidence can be tested, a check is made to insure that no block transfer, exchange sequence, instruction fetch sequence, or scalar CP2 is in progress. If so, the I/O request is blocked and must be resubmitted eight clock periods later. The lower four bits[†] of an I/O reference are tested against ranks A, B, and C. Coincidence with rank A, B, or C disallows the I/O request. These ranks may be holding previous scalar or I/O memory requests. An I/O request that is disallowed must wait eight clock periods before it can request again.

[†] Three bits for 8-bank phasing; see description in section 5.

I/O MEMORY REQUEST CONDITIONS

The following conditions must be present for a memory request to be processed:

1. I/O request

2. No coincidence in rank A, B, or C

3. No scalar instruction in clock period two of a scalar sequence

4. No fetch request

5. No 176, 177, or 034 through 037 process

6. No exchange sequence

7. No 033 request

I/O MEMORY ADDRESSING

All I/O memory references are absolute. The current and limit registers are 20 bits, allowing I/O access to all of memory. Setting of the current and limit registers is limited to monitor mode.

REAL-TIME CLOCK

Programs can be timed precisely by using the clock period counter. This counter is advanced one count each clock period of 12.5 nanoseconds. Since the clock is advanced synchronously with program execution, it may be used to time the program to an exact number of clock periods.

Instructions used with the real-time clock are:

    0014j0   Enter the real-time clock register with (Sj)
    072ixx   Transmit (RTC) to Si

The clock period counter is a 64-bit counter that can be read by a program through the use of the 072 instruction and can be reset only by the 0014j0 monitor instruction.

PROGRAMMABLE CLOCK OPTION

Cray Research provides as a standard option a programmable clock thay may
be used tc measure the duration of intervals accurately. A periodic inter-
rupt can be generated with intervals selected under monitor program control.
The clock frequency is 80 Mhz. Intervals from 12.5 nanoseconds to
53.7 seconds are possible; however, intervals shorter than about 100
microseconds are not practical due to the monitor overhead involved in
processing the interrupt.

INSTRUCTIONS

Provided with the clock are four additional instructions made possible by
redefining the k designator for the 0014 instruction. The option also
makes available two additonal registers: the interrupt interval register
(II) and the interrupt countdown counter (ICD).

0014j4	Enter interrupt interval (II) register with (Sj)
0014j5	Clear the programmable clock interrupt request
0014j6	Enable the programmable clock interrupt request
0014j7	Disable the programmable clock interrupt requests

INTERRUPT INTERVAL REGISTER

The interrupt interval (II) register is a 32-bit register that can be
loaded with a binary value equal to the number of clock periods that are
to elapse between programmable clock interrupt requests. The interrupt
interval is transferred from the lower 32 bits of the Sj register into
both the interrupt interval register and the interrupt countdown (ICD)
counter when the 0014j4 instruction is executed. This interval value is
held in the register and repeatedly sampled by the interrupt countdown
counter until another 0014j4 instruction is received to change the interval
value.

INTERRUPT COUNTDOWN COUNTER

The interrupt countdown (ICD) counter is a 32-bit counter that is preset
to the contents of the interrupt interval register when the 0014j4 instruc-
tion is executed. This counter runs continuously but counts down, decrement-
ing by one each clock period until the contents of the counter are zero. At
this time, it sets the programmable clock interrupt request. The counter
then samples the interval value held in the interrupt interval register and
repeats the countdown to zero cycle, setting the programmable clock
interrupt request at regular intervals determined by the interval value.
When the programmable clock interrupt request is set, it remains set
until a 0014j5 instruction, clear programmable clock interrupt request,
is executed. A programmable clock interrupt request can be set only after
the 0014j6 instruction has been executed to enable the interrupt. A pro-
grammable clock interrupt request only causes an interrupt when not in
monitor mode; a request set in monitor mode is held until the system
switches to user mode.

CLEAR PROGRAMMABLE CLOCK INTERRUPT REQUEST

Following a program interrupt interval, an active programmable clock
interrupt request may be cleared by executing the 0014j5 clear program-
mable clock interrupt instruction.

Following any deadstart, the monitor program should insure the state of
the programmable clock interrupt by clearing programmable clock interrupt
requests (0014j5) and disabling programmable clock interrupt requests (0014j7).

APPENDIX SECTION

When issue conditions are satisfied an instruction completes in a fixed amount of time. Instruction issue may cause reservations to be placed on a functional unit or registers. Knowledge of the issue conditions, instruction execution times and reservations permit accurate timing of code sequences. Memory bank conflicts due to I/O activity are the only element of unpredictability.

SCALAR INSTRUCTIONS

Four conditions must be satisfied for issue of a scalar instruction:

1. The functional unit must be free. No conflicts can arise with other scalar instructions; however, vector floating point instructions reserve the floating point units. Memory references may be delayed due to conflicts.

2. The result register must be free.

3. The operand register must be free.

4. Issue is delayed 1 clock period if a result register group input path conflict would exist with a previously issued instruction. One input path exists for each of the four register groups (A, B, S and T).

Scalar instructions place reservations only on result registers. A result register is reserved for the execution time of the instruction. No reservations are placed on the functional unit or operand registers.

A transmit vector mask to Si (073) instruction is delayed by (VL) + 6 clock periods from the issue of a previous vector mask (175) instruction and is delayed by 6 clock periods from the issue of a preceding transmit (Sj) to VM (003) instruction.

Execution times in clock periods are given below.

(A=A register, M=Memory, B=B register, S=S register, I=Immediate, C=Channel)

24-bit results:

A ← M	11*	A ← C	4
M ← A	1*	A ← A+A	2
A ← B	1	A ← AxA	6
B ← A	1	A ← pop(S)	4
A ← S	1	A ← lzc(S)	3
A ← I	1	VL ← A	1

64-bit results:

S ← M	11*	S ← S+S	3
M ← S	1*	S ← S(f.add)S	6*
S ← T	1	S ← S(f.mult)S	7*
T ← S	1	S ← S(r.a.)	14*
S ← I	1	S ← V	5
S ← S(log.)S	1	V ← S	3
S ← S(shift)I	2	S ← VM	1
S ← S(shift)A	3	S ← RTC	1
S ← S(mask)I	1	S ← A	2
RTC ← S	1	VM ← S	3

* Issue may be delayed because of a functional unit reservation by a
vector instruction. Memory may be considered a functional unit for
timing considerations.

VECTOR INSTRUCTIONS

Four conditions must be satisfied for issue of a vector instruction:

1. The functional unit must be free. (Conflicts may occur with vector
    operations.)

2. The result register must be free. (Conflicts may occur with vector
    operations.)

3. The operand registers must be free or at chain slot time.

4. Memory must be quiet if the instruction references memory.

Vector instructions place reservations on functional units and registers
for the duration of execution.

1. Functional units are reserved for VL+4 clock periods. Memory is
    reserved for VL+5 clock periods on a write operation, VL+4 clock
    periods on a read operation.

2. The result register is reserved for the functional unit time
    +(VL+2) clock periods. The result register is reserved for the
    functional unit +7 clock periods if the vector length is less than
    5. At functional unit time +2 (chain slot time) a subsequent
    instruction, which has met all other issue conditions, may issue. This
    process is called "chaining." Several instructions using different
    functional units may be chained in this manner to attain a significant
    enhancement of processing speed.

3. Vector operand registers are reserved for VL clock periods. Vector
    operand registers are reserved for 5 clock periods if the vector
    length is less than 5. The vector register used in a block store to
    memory (177 instruction) is reserved for VL clock periods. Scalar
    operand registers are not reserved.

Vector instructions produce one result per clock period. The functional
unit times are given below. The vector read and write instructions
(176, 177) produce results more slowly if bank conflicts arise due to
the increment value (Ak) being a multiple of 8[†]. Chaining cannot occur
for the vector read operation in this case.

If (Ak) is an odd multiple of 8[†], results are produced every 2 clock
periods.

If (Ak) is an even multiple of 8[†], results are produced every 4 clock
periods.

Functional unit	Time (c.p.)
Logical	2
Shift	4
Integer add	3
Floating add	6
Floating multiply	7
Reciprocal approximation	14
Memory	7

† Multiple of 4 for 8-bank phasing; refer to section 5.

Memory must be quiet before issue of the B and T register block copy instructions (034-037). Subsequent instructions may not issue for 14+ (Ai) clock periods if (Ai)≠0 and 5 clock periods if (Ai)=0 when reading data to the B and T registers (034,036). They may not issue for 6+(Ai) clock periods when storing data (035,037).

The B and T register block read (034,036) instructions require that there be no register reservation on the A and S registers, respectively, before issue.

Branch instructions cannot issue until an A0 or S0 operand register has been free for one clock period. Fall-through in buffer requires two clock periods. Branch-in-buffer requires five clock periods. When an "out of buffer" condition occurs the execution time for a branch instruction is 14 clock periods.[†]

A two parcel instruction takes two clock periods to issue.

Instruction issue is delayed 2 clock periods when the next instruction parcel is in a different instruction parcel buffer. Instruction issue is delayed 14 clock periods if the next instruction parcel is not in an instruction parcel buffer.

HOLD MEMORY

A delay of 1, 2, or 3 CP will be added to a scalar memory read if a bank conflict occurs with rank C, B, or A, respectively, of the memory access network. A conflict occurs if the address is in the same bank as the address in rank C, B, or A. Conflicts can occur only with scalar or I/O references. The scalar instruction senses the conflict condition at issue time + 1 CP. The scalar instruction address enters rank A of the memory access network at issue time + 1 CP. The scalar instruction address enters rank B at issue + 2 CP. The scalar instruction address enters rank C at issue + 3 CP.

---

† 18 clock periods for 8-bank phasing option; refer to section 5.

Scalar instruction timing (no conflict):

CP n	Issue, reserve register
CP n+1	Address rank A, sense conflict
CP n+2	Address rank B
CP n+3	Address rank C
⋮	
CP n+9	Clear register reservation
CP n+10	Issue

HOLD ISSUE

A delay of issue results if a 100 - 137 instruction is in the NIP register and a hold memory condition exists. The delay will depend on the hold memory delay.

A delay of issue results if a 100 - 137 instruction is in the NIP register and a 100 - 137 instruction in process senses a conflict with rank A, B, or C.

An additional 1 CP delay is added to a hold memory condition if a 070 instruction destination register conflict is sensed.
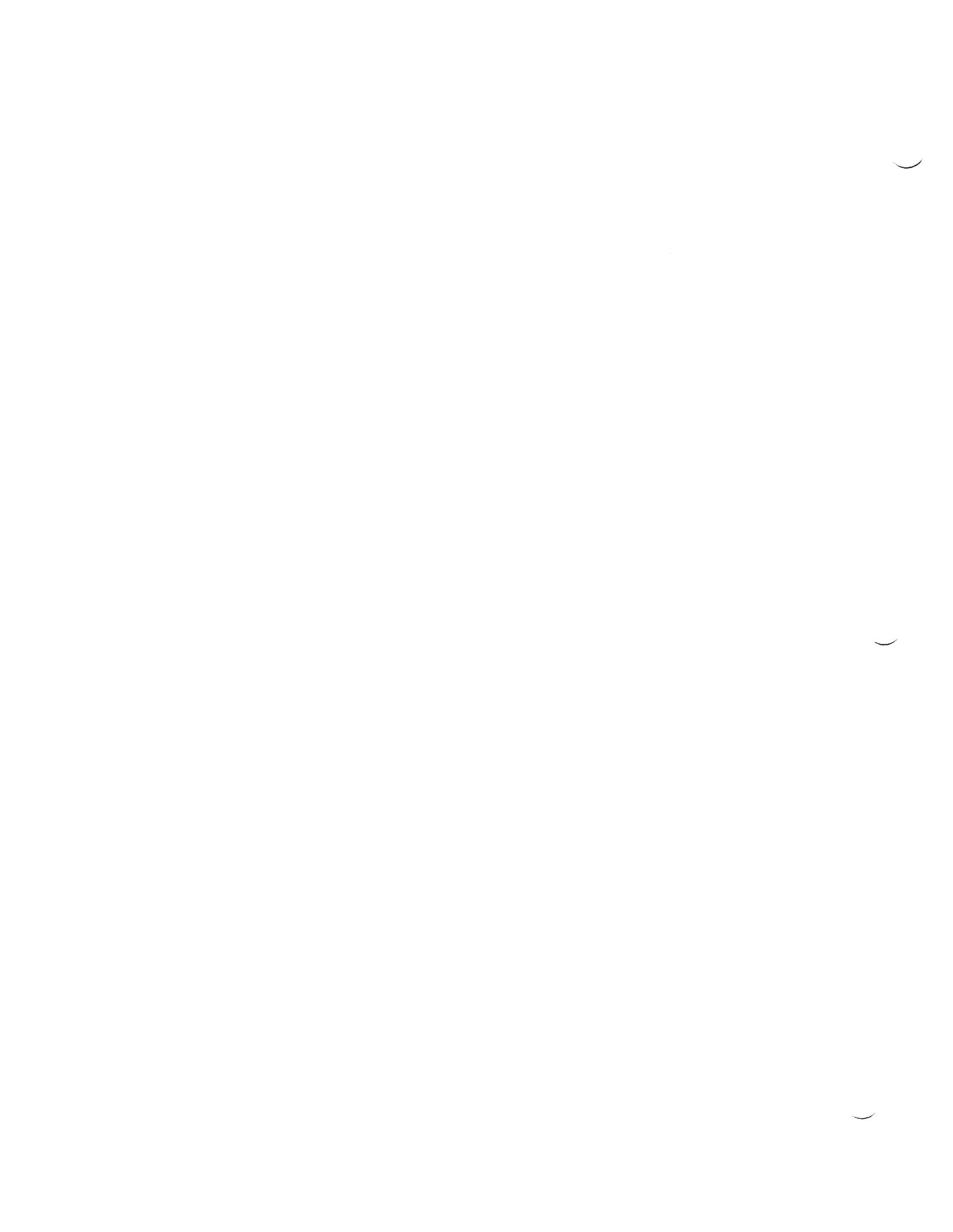
Alpha Code	Application	No. Used	Alpha Code	Application	No. Used
	A SERIES MODULES			G SERIES MODULES	
AA	Address adder	5	GA	Scalar single shift	4
AB	Storage block address	2	GB	Scalar double shift (front half)	4
AC	Vector storage control	1	GC	Scalar double shift (back half)	4
AD	Storage address distribution	3	GD	Data Ak to Si extended	1
AE	B and T storage control	1	GE	Scalar add (front half)	4
AF	Address multiply levels 1 and 2	3	GF	Scalar add (back balf)	2
AG	Address multiply level 2	3	GG	Constant to Si	1
AH	Address multiply upper level 3	1	GH	Pop and zero count to Ai	1
AI	Address multiply lower level 3	1	GI	Real time clock	2
AJ	Address multiply level 4	1	GJ*	RTC/PCI (lower bits)	1
AR	Address registers	12	GK*	RTC/PCI (upper bits)	1
	D SERIES MODULES		GR	Scalar registers	32
DE	Address merge fanout	10		H SERIES MODULES	
DF	Channel reference control	1	HA	Program branch control	1
DG	Channel interrupt control	1	HB	Next instruction parcel	4
DH	Channel address control	1	HC	Lower program address	1
DI	Synchronizing circuits	3	HD	Upper program address	2
DJ	Input channel control 16-bit	++	HE	Program parameter data	4
DK	Output channel control 16-bit	++	HF	Fetch sequence control	1
DL	Input data assembly 16-bit	12	HS	Instruction buffers	8
DM	Output data disassembly 16-bit	12	HX	Exchange sequence control	1
DN	Input channel control	++		J SERIES MODULES	
DO	Output channel control	++	JA	CIP fanout to AR modules	5
DU	Input channel control	+	JB	CIP fanout to GR modules	10
DV	Input channel control	+	JC	Select vector data paths	1
DZ	Unused I/O channel termination	++	JD	Vector function issue control	1
	F SERIES MODULES		JE	Floating point issue control	1
FA	Floating add exponent input operands	1	JF	Vector register issue control	1
FB	Floating add exponent input operands	1	JG	Scalar register issue control	1
FC	Floating add coefficient input operands	4	JH	Address register issue control	1
FD	Floating add coefficient alignment	4	JI	Storage access issue control	1
FE	Floating add coefficient add (front half)	3	JJ	Hold storage issue control	1
FF	Floating add coefficient add (back half)	3	JK	Address access control	1
FG	Floating add coefficient result	2	JL	Scalar access control	1
FH	Floating add coefficient result	1			
FI	Floating add exponent data	1			
FJ	Floating add exponent result	1			

* When the Programmable Clock Option is installed, a GJ module and a GK module replace the two GI modules.

† DU, DV modules are used to communicate with various CRI interfaces. The number of modules varies with the system configuration.

†† The number of modules depends on the configuration.

Alpha Code	Application	No. Used		Alpha Code	Application	No. Used
	M SERIES MODULES				**T SERIES MODULES**	
MA	First level product	24		TC	Clock fanout	9
MB	Second level product	10		TO	Master clock	1
MC	Third level product	8		TX**	16-bank phasing	2
MD	Fourth level product	3		TY**	8-bank phasing	2
ME	Fifth level product	3		TZ	Master clock fanout	1
MF	First level ends	2			**V SERIES MODULES**	
MG	First section exponents	1		VA	Data to vector registers	32
MH	Last section exponents	1		VB	Vector data to jk functions	32
	R SERIES MODULES			VC	Vector data to j functions	16
RA	Table for Ao	1		VD	Vector length control	1
RB	Table for Ao^2	1		VE	Vector write control	1
RC	Form A$_1$	3		VF	Front half vector shift	4
RD	Form A$_1$	1		VG	Back half vector shift	4
RE	Form A$_1$	1		VH	Front half vector add	4
RF	Form A$_1$	2		VI	Back half vector add	2
RG	Form A$_1$	1		VJ	Vector logical data	4
RH	Form A$_{1_2}^2$	2		VK	Vector logical control	1
RI	Form A$_{1_2}^2$	1		VL	Vector Pop Count Option	1†
RJ	Form A$_{1_2}^2$	1		VR	Vector registers	32
RK	Form A$_{1_2}^2$	2			**Z SERIES MODULES***	
RL	Form A$_{1_2}^2$	1		ZB	Storage w/memory data buffers	288
RM	Form A$_2$	10		ZC	Storage with clock fanout	36
RN	Form A$_2$	2		ZD	Storage R/W control	1
RO	Form A$_2$	7		ZE	Storage section control	2
RP	Form A$_2$	1		ZF	Storage with address fanout	120
RQ	Form A$_2$	3		ZG	Check bit generation	2
RR	Form A$_2$	1		ZI	Corrective storage	1
RS	Reciprocal coefficient	2		ZK	Syndrome generation and error correction	32
RT	Reciprocal coefficient	2		ZY	Storage module	120
RU	Operand delay	9		ZZ	Storage module w/address fan-out	588
RV	Result exponent	1				
	S SERIES MODULES					
SH*	16-bit synchronous input data assembly					
SI*	16-bit synchronous output data assembly					

---

* One SH, SI module pair interfaces to each CRI disk controller. The number depends on the system configuration.

** For 8-bank phasing, TY modules are substituted for TX modules.

*** Figures are for 16-bank memory.

$\dagger$ Included when Vector Population Count Option is present.

References to software in this publication are limited to those features
of the hardware that provide for software or take it into consideration.

SYSTEM MONITOR

A monitor program is loaded at system dead start and remains in memory
for as long as the system is used. Only the monitor program executes
in monitor mode and can execute monitor instructions. A program
executing in monitor mode cannot be interrupted unless the Monitor Mode
Interrupt (MMI) option is present. A monitor program is designed to
reference all of memory.

OBJECT PROGRAM

An object program as referred to in this publication means any program
other than the monitor program. Generally, the term describes a job-
oriented program but may also describe an operating system task that does
not execute in monitor mode. An object program may be a machine language
program such as a FORTRAN compiler or it may be a program resulting from
compilation of FORTRAN statements by the compiler.

OPERATING SYSTEM

The operating system consists of a monitor program, object programs that
perform system-related functions, compilers, assemblers, and various
utility programs. The operating system is loaded into memory and possibly
onto mass storage during system dead start. Features of the operating system
system and organization of storage, which is a function of the operating
system, will be described in the operating system reference manual.

SYSTEM OPERATION

System operation begins at CPU dead start. Dead start is that sequence of
operations required to start a program running in the computer after power
has been turned off and then turned on again.

The dead start sequence is initiated from the maintenance control unit
(MCU). The sequence is described in detail in Section 3. During the
dead start sequence, the MCU loads a program containing an exchange
package at absolute address zero in the CRAY-1 memory. A signal from
the MCU causes the CRAY-1 to begin execution of the program pointed to by
the exchange package.

FLOATING POINT RANGE ERRORS

Detection of the floating point range error initiates an interrupt if the
floating point mode flag is set in the mode register and monitor mode is
not in effect. The programmer has the capability via the 0022 instruction
to clear the floating point mode flag so that results going out of range
are prevented from interrupting. This is especially useful for operations
such as the vector merge instruction usage in subroutines such as SINE and
COSINE, where some results may be known to go out of range. At the end
of the code sequence, the programmer normally resets the floating point
mode via a 0021 instruction.

INSTRUCTION SUMMARY

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
000xxx	ERR		4-7	-	Error exit
†000ijk	ERR	exp	4-7	-	Error exit
††0010jk	CA,Aj	Ak	4-8	-	Set the channel (Aj) current address to (Ak) and begin the I/O sequence
††0011jk	CL,Aj	Ak	4-8	-	Set the channel (Aj) limit address to (Ak)
††0012jx	CI,Aj		4-8	-	Clear channel (Aj) interrupt flag
††0013jx	XA	Aj	4-8	-	Enter XA register with (Aj)
††0014j0	RT	Sj	4-10	-	Enter RTC register with (Sj)
††§0014j4	PCI	Sj	4-10	-	Enter interval register with (Sj)
††§0014j5	CCI		4-10	-	Clear PCI request
††§0014j6	ECI		4-10	-	Enable PCI request
††§0014j7	DCI		4-10	-	Disable PCI request
0020xk	VL	Ak	4-12	-	Transmit (Ak) to VL register
†0020x0	VL	1	4-12	-	Transmit 1 to VL register
0021xx	EFI		4-13	-	Enable interrupt on floating point error
0022xx	DFI		4-13	-	Disable interrupt on floating point error
003xjx	VM	Sj	4-14	-	Transmit (Sj) to VM register
†003x0x	VM	0	4-14	-	Clear VM register
004xxx	EX		4-15	-	Normal exit
†004ijk	EX	exp	4-15	-	Normal exit
005xjk	J	Bjk	4-16	-	Jump to (Bjk)
006ijkm	J	exp	4-17	-	Jump to exp
007ijkm	R	exp	4-18	-	Return jump to exp; set B00 to P
010ijkm	JAZ	exp	4-19	-	Branch to exp if (A0) = 0
011ijkm	JAN	exp	4-19	-	Branch to exp if (A0) ≠ 0
012ijkm	JAP	exp	4-19	-	Branch to exp if (A0) positive
013ijkm	JAM	exp	4-19	-	Branch to exp if (A0) negative
014ijkm	JSZ	exp	4-20	-	Branch to exp if (S0) = 0
015ijkm	JSN	exp	4-20	-	Branch to exp if (S0) ≠ 0
016ijkm	JSP	exp	4-20	-	Branch to exp if (S0) positive
017ijkm	JSM	exp	4-20	-	Branch to exp if (S0) negative
020ijkm			4-21	-	Transmit exp = jkm to Ai
021ijkm	Ai	exp	4-21	-	Transmit exp = 1's complement of jkm to Ai
022ijk			4-22	-	Transmit exp = jk to Ai
023ijx	Ai	Sj	4-23	-	Transmit (Sj) to Ai
024ijk	Ai	Bjk	4-24	-	Transmit (Bjk) to Ai
025ijk	Bjk	Ai	4-24	-	Transmit (Ai) to Bjk
026ij0	Ai	PSj	4-25	Pop/LZ	Population count of (Sj) to Ai
§§026ij1	Ai	QSj	4-25	Pop/LZ	Population count parity of (Sj) to Ai
027ijx	Ai	ZSj	4-26	Pop/LZ	Leading zero count of (Sj) to Ai
030ijk	Ai	Aj+Ak	4-27	A Int Add	Integer sum of (Aj) and (Ak) to Ai
†030i0k	Ai	Ak	4-27	A Int Add	Transmit (Ak) to Ai
†030ij0	Ai	Aj+1	4-27	A Int Add	Integer sum of (Aj) and 1 to Ai
031ijk	Ai	Aj-Ak	4-27	A Int Add	Integer difference of (Aj) less (Ak) to Ai

† Special syntax form

†† Privileged to monitor mode

§ Programmable Clock Option only

§§ Vector Population Count Option only

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
†031i00	Ai	-1	4-27	A Int Add	Transmit -1 to Ai
†031i0k	Ai	-Ak	4-27	A Int Add	Transmit the negative of (Ak) to Ai
†031ij0	Ai	Aj-1	4-27	A Int Add	Integer difference of (Aj) less 1 to Ai
032ijk	Ai	Aj*Ak	4-28	A Int Mult	Integer product of (Aj) and (Ak) to Ai
033i0x	Ai	CI	4-29	-	Channel number to Ai (j=0)
033ij0	Ai	CA,Aj	4-29	-	Address of channel (Aj) to Ai (j≠0; k=0)
033ij1	Ai	CE,Aj	4-29	-	Error flag of channel (Aj) to Ai (j≠0; k=1)
034ijk	Bjk,Ai	,A0	4-31	Memory	Read (Ai) words to B register jk from (A0)
†034ijk	Bjk,Ai	0,A0	4-31	Memory	Read (Ai) words to B register jk from (A0)
035ijk	,A0	Bjk,Ai	4-31	Memory	Store (Ai) words at B register jk to (A0)
†035ijk	0,A0	Bjk,Ai	4-31	Memory	Store (Ai) words at B register jk to (A0)
036ijk	Tjk,Ai	,A0	4-31	Memory	Read (Ai) words to T register jk from (A0)
†036ijk	Tjk,Ai	0,A0	4-31	Memory	Read (Ai) words to T register jk from (A0)
037ijk	,A0	Tjk,Ai	4-31	Memory	Store (Ai) words at T register jk to (A0)
†037ijk	0,A0	Tjk,Ai	4-31	Memory	Store (Ai) words at T register jk to (A0)
040ijkm	Si	exp	4-33	-	Transmit jkm to Si
041ijkm			4-33	-	Transmit exp = 1's complement of jkm to Si
042ijk	Si	<exp	4-34	S Logical	Form 1's mask exp = 64-jk bits in Si from the right
	Si	#>exp	4-34		
†042i77	Si	1	4-34	S Logical	Enter 1 into Si
†042i00	Si	-1	4-34	S Logical	Enter -1 into Si
043ijk	Si	>exp	4-34	S Logical	Form 1's mask exp = jk bits in Si from the left
	Si	#<exp	4-34		
†043i00	Si	0	4-34	S Logical	Clear Si
044ijk	Si	Sj&Sk	4-35	S Logical	Logical product of (Sj) and (Sk) to Si
†044ij0	Si	Sj&SB	4-35	S Logical	Sign bit of (Sj) to Si
†044ij0	Si	SB&Sj	4-35	S Logical	Sign bit of (Sj) to Si (j≠0)
045ijk	Si	#Sk&Sj	4-35	S Logical	Logical product of (Sj) and 1's complement of (Sk) to Si
†045ij0	Si	#SB&Sj	4-35	S Logical	(Sj) with sign bit cleared to Si
046ijk	Si	Sj\Sk	4-35	S Logical	Logical difference of (Sj) and (Sk) to Si
†046ij0	Si	Sj\SB	4-35	S Logical	Toggle sign bit of Sj, then enter into Si
†046ij0	Si	SB\Sj	4-35	S Logical	Toggle sign bit of Sj, then enter into Si (j≠0)
047ijk	Si	#Sj\Sk	4-35	S Logical	Logical equivalence of (Sk) and (Sj) to Si
†047i0k	Si	#Sk	4-35	S Logical	Transmit 1's complement of (Sk) to Si
†047ij0	Si	#Sj\SB	4-35	S Logical	Logical equivalence of (Sj) and sign bit to Si
†047ij0	Si	#SB\Sj	4-35	S Logical	Logical equivalence of (Sj) and sign bit to Si (j≠0)
†047i00	Si	#SB	4-35	S Logical	Enter 1's complement of sign bit into Si
050ijk	Si	Sj!Si&Sk	4-35	S Logical	Logical product of (Si) and (Sk) complement ORed with logical product of (Sj) and (Sk) to Si
†050ij0	Si	Sj!Si&SB	4-35	S Logical	Scalar merge of (Si) and sign bit of (Sj) to Si
051ijk	Si	Sj!Sk	4-35	S Logical	Logical sum of (Sj) and (Sk) to Si
†051i0k	Si	Sk	4-35	S Logical	Transmit (Sk) to Si
†051ij0	Si	Sj!SB	4-35	S Logical	Logical sum of (Sj) and sign bit to Si
†051ij0	Si	SB!Sj	4-35	S Logical	Logical sum of (Sj) and sign bit to Si (j≠0)
†051i00	Si	SB	4-35	S Logical	Enter sign bit into Si
052ijk	S0	Si<exp	4-38	S Shift	Shift (Si) left exp = jk places to S0
053ijk	S0	Si>exp	4-38	S Shift	Shift (Si) right exp = 64-jk places to S0
054ijk	Si	Si<exp	4-38	S Shift	Shift (Si) left exp = jk places
055ijk	Si	Si>exp	4-38	S Shift	Shift (Si) right exp = 64-jk places
056ijk	Si	Si,Sj<Ak	4-39	S Shift	Shift (Si and Sj) left (Ak) places to Si
†056ij0	Si	Si,Sj<1	4-39	S Shift	Shift (Si and Sj) left one place to Si
†056i0k	Si	Si<Ak	4-39	S Shift	Shift (Si) left (Ak) places to Si

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
057ijk	Si	Sj,Si>Ak	4-39	S Shift	Shift (Sj and Si) right (Ak) places to Si
†057ij0	Si	Sj,Si>1	4-39	S Shift	Shift (Sj and Si) right one place to Si
†057i0k	Si	Si>Ak	4-39	S Shift	Shift (Si) right (Ak) places to Si
060ijk	Si	Sj+Sk	4-40	S Int Add	Integer sum of (Sj) and (Sk) to Si
061ijk	Si	Sj-Sk	4-40	S Int Add	Integer difference of (Sj) and (Sk) to Si
†061i0k	Si	-Sk	4-40	S Int Add	Transmit negative of (Sk) to Si
062ijk	Si	Sj+FSk	4-41	F.P. Add	Floating sum of (Sj) and (Sk) to Si
†062i0k	Si	+FSk	4-41	F.P. Add	Normalize (Sk) to Si
063ijk	Si	Sj-FSk	4-41	F.P. Add	Floating difference of (Sj) and (Sk) to Si
†063i0k	Si	-FSk	4-41	F.P. Add	Transmit normalized negative of (Sk) to Si
064ijk	Si	Sj*FSk	4-42	F.P. Mult	Floating product of (Sj) and (Sk) to Si
065ijk	Si	Sj*HSk	4-42	F.P. Mult	Half precision rounded floating product of (Sj) and (Sk) to Si
066ijk	Si	Sj*RSk	4-42	F.P. Mult	Full precision rounded floating product of (Sj) and (Sk) to Si
067ijk	Si	Sj*ISk	4-42	F.P. Mult	2 - Floating product of (Sj) and (Sk) to Si
070ijx	Si	/HSj	4-44	F.P. Rcpl	Floating reciprocal approximation of (Sj) to Si
071i0k	Si	Ak	4-45	-	Transmit (Ak) to Si with no sign extension
071i1k	Si	+Ak	4-45	-	Transmit (Ak) to Si with sign extension
071i2k	Si	+FAk	4-45	-	Transmit (Ak) to Si as unnormalized floating point number
071i3x	Si	0.6	4-45	-	Transmit constant 0.75*2**48 to Si
071i4x	Si	0.4	4-45	-	Transmit constant 0.5 to Si
071i5x	Si	1.	4-45	-	Transmit constant 1.0 to Si
071i6x	Si	2.	4-45	-	Transmit constant 2.0 to Si
071i7x	Si	4.	4-45	-	Transmit constant 4.0 to Si
072ixx	Si	RT	4-47	-	Transmit (RTC) to Si
073ixx	Si	VM	4-47	-	Transmit (VM) to Si
074ijk	Si	Tjk	4-47	-	Transmit (Tjk) to Si
075ijk	Tjk	Si	4-47	-	Transmit (Si) to Tjk
076ijk	Si	Vj,Ak	4-48	-	Transmit (Vj, element (Ak)) to Si
077ijk	Vi,Ak	Sj	4-48	-	Transmit (Sj) to Vi element (Ak)
†077i0k	Vi,Ak	0	4-48	-	Clear Vi element (Ak)
10hijkm	Ai	exp,Ah	4-49	Memory	Read from ((Ah) + exp) to Ai (A0=0)
†100ijkm	Ai	exp,0	4-49	Memory	Read from (exp) to Ai
†100ijkm	Ai	exp,	4-49	Memory	Read from (exp) to Ai
†10hi000	Ai	,Ah	4-49	Memory	Read from (Ah) to Ai
11hijkm	exp,Ah	Ai	4-49	Memory	Store (Ai) to (Ah) + exp (A0=0)
†110ijkm	exp,0	Ai	4-49	Memory	Store (Ai) to exp
†110ijkm	exp,	Ai	4-49	Memory	Store (Ai) to exp
†11hi000	,Ah	Ai	4-49	Memory	Store (Ai) to (Ah)
12hijkm	Si	exp,Ah	4-49	Memory	Read from ((Ah) + exp) to Si (A0=0)
†120ijkm	Si	exp,0	4-49	Memory	Read from (exp) to Si
†120ijkm	Si	exp,	4-49	Memory	Read from (exp) to Si
†12hi000	Si	,Ah	4-49	Memory	Read from (Ah) to Si
13hijkm	exp,Ah	Si	4-49	Memory	Store (Si) to (Ah) + exp (A0=0)
†130ijkm	exp,0	Si	4-49	Memory	Store (Si) to exp
†130ijkm	exp,	Si	4-49	Memory	Store (Si) to exp
†13hi000	,Ah	Si	4-49	Memory	Store (Si) to (Ah)
140ijk	Vi	Sj&Vk	4-51	V Logical	Logical products of (Sj) and (Vk) to Vi
141ijk	Vi	Vj&Vk	4-51	V Logical	Logical products of (Vj) and (Vk) to Vi
142ijk	Vi	Sj!Vk	4-51	V Logical	Logical sums of (Sj) and (Vk) to Vi
†142i0k	Vi	Vk	4-51	V Logical	Transmit (Vk) to Vi

---

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
143ijk	Vi	Vj!Vk	4-51	V Logical	Logical sums of (Vj) and (Vk) to Vi
144ijk	Vi	Sj\Vk	4-51	V Logical	Logical differences of (Sj) and (Vk) to Vi
145ijk	Vi	Vj\Vk	4-51	V Logical	Logical differences of (Vj) and (Vk) to Vi
†145iii	Vi	0	4-51	**V Logical**	**Clear Vi**
146ijk	Vi	Sj!Vk&VM	4-51	**V Logical**	**Transmit (Sj) if VM bit = 1; (Vk) if VM bit = 0 to Vi**
†146i0k	Vi	#VM&Vk	4-51	V Logical	Vector merge of (Vk) and 0 to Vi
147ijk	Vi	Vj!Vk&VM	4-51	V Logical	Transmit (Vj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
150ijk	Vi	Vj<Ak	4-55	V Shift	Shift (Vj) left (Ak) places to Vi
†150ij0	Vi	Vj<1	4-55	V Shift	Shift (Vj) left one place to Vi
151ijk	Vi	Vj>Ak	4-55	V Shift	Shift (Vj) right (Ak) places to Vi
†151ij0	Vi	Vj>1	4-55	V Shift	Shift (Vj) right one place to Vi
152ijk	Vi	Vj,Vj<Ak	4-56	V Shift	Double shift (Vj) left (Ak) places to Vi
†152ij0	Vi	Vj,Vj<1	4-56	V Shift	Double shift (Vj) left one place to Vi
153ijk	Vi	Vj,Vj>Ak	4-56	V Shift	Double shift (Vj) right (Ak) places to Vi
153ij0	Vi	Vj,Vj>1	4-56	V Shift	Double shift (Vj) right one place to Vi
154ijk	Vi	Sj+Vk	4-61	V Int Add	Integer sums of (Sj) and (Vk) to Vi
155ijk	Vi	Vj+Vk	4-61	V Int Add	Integer sums of (Vj) and (Vk) to Vi
156ijk	Vi	Sj-Vk	4-61	V Int Add	Integer differences of (Sj) and (Vk) to Vi
†156i0k	Vi	-Vk	4-61	V Int Add	Transmit negative of (Vk) to Vi
157ijk	Vi	Vj-Vk	4-61	V Int Add	Integer differences of (Vj) and (Vk) to Vi
160ijk	Vi	Sj*FVk	4-63	F.P. Mult	Floating products of (Sj) and (Vk) to Vi
161ijk	Vi	Vj*FVk	4-63	F.P. Mult	Floating products of (Vj) and (Vk) to Vi
162ijk	Vi	Sj*HVk	4-63	F.P. Mult	Half precision rounded floating products of (Sj) and (Vk) to Vi
163ijk	Vi	Vj*HVk	4-63	F.P. Mult	Half precision rounded floating products of (Vj) and (Vk) to Vi
164ijk	Vi	Sj*RVk	4-63	F.P. Mult	Rounded floating products of (Sj) and (Vk) to Vi
165ijk	Vi	Vj*RVk	4-63	F.P. Mult	Rounded floating products of (Vj) and (Vk) to Vi
166ijk	Vi	Sj*IVk	4-63	F.P. Mult	2 - floating products of (Sj) and (Vk) to Vi
167ijk	Vi	Vj*IVk	4-63	F.P. Mult	2 - floating products of (Vj) and (Vk) to Vi
170ijk	Vi	Sj+FVk	4-66	F.P. Add	Floating sums of (Sj) and (Vk) to Vi
†170i0k	Vi	+FVk	4-66	F.P. Add	Normalize (Vk) to Vi
171ijk	Vi	Vj+FVk	4-66	F.P. Add	Floating sums of (Vj) and (Vk) to Vi
172ijk	Vi	Sj-FVk	4-66	F.P. Add	Floating differences of (Sj) and (Vk) to Vi
†172i0k	Vi	-FVk	4-66	F.P. Add	Transmit normalized negatives of (Vk) to Vi
173ijk	Vi	Vj-FVk	4-66	F.P. Add	Floating differences of (Vj) and (Vk) to Vi
174ij0	Vi	/HVj	4-68	F.P. Rcpl	Floating reciprocal approximations of (Vj) to Vi
§§174ij1	Vi	PVi	4-70	F.P. Rcpl	Population counts of (Vj) to Vi
§§174ij2	VI	QVi	4-70	F.P. Rcpl	Population count parities of (Vj) to Vi
175xj0	VM	Vj,Z	4-71	V Logical	VM=1 where (Vj) = 0
175xj1	VM	Vj,N	4-71	V Logical	VM=1 where (Vj) ≠ 0
175xj2	VM	Vj,P	4-71	V Logical	VM=1 where (Vj) positive
175xj3	VM	Vj,M	4-71	V Logical	VM=1 where (Vj) negative
176ixk	Vi	,A0,Ak	4-73	Memory	Read (VL) words to Vi from (A0) incremented by (Ak)
†176ix0	Vi	,A0,1	4-73	Memory	Read (VL) words to Vi from (A0) incremented by 1
177xjk	,A0,Ak	Vj	4-73	Memory	Store (VL) words from Vj to (A0) incremented by (Ak)
†177xj0	,A0,1	Vj	4-73	Memory	Store (VL) words from Vj to (A0) incremented by 1

† Special syntax form

§§ Vector Population Count Option only

READERS COMMENT FORM

CRAY-1 Hardware Reference Manual   HR-0004 F

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME _____

JOB TITLE _____

FIRM_____

ADDRESS_____

CITY _____ STATE _____ ZIP _____

CRAY
RESEARCH, INC.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS   PERMIT NO. 6184   ST PAUL MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN   55120
U.S.A.

STAPLE

READERS COMMENT FORM

CRAY-1 Hardware Reference Manual

HR-0004 F

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME _____

JOB TITLE _____

FIRM_____

ADDRESS_____

CITY_____STATE _____ ZIP _____

CRAY
RESEARCH, INC.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS   PERMIT NO. 6184   ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

STAPLE