CRAY RESEARCH, INC.

PRELIMINARY
CAL ASSEMBLER
REFERENCE MANUAL

CRAY-1 COMPUTER

THE CRAY-1 COMPUTER

PRELIMINARY

CAL ASSEMBLER

REFERENCE MANUAL

COPYRIGHT (C) 1975 BY CRAY RESEARCH, INC. THIS MANUAL, OR PARTS THEREOF, MAY NOT BE REPRODUCED IN ANY FORM WITHOUT PERMISSION OF CRAY RESEARCH, INC. THIS MANUAL CONTAINS PRELIMINARY INFORMATION WHICH IS SUBJECT TO CORRECTION AND CHANGE WITHOUT FORMAL NOTICE TO COPY HOLDERS.

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

REVISION RECORD

REVISION	DESCRIPTION
4/75	Original Printing
A	This revision corrects various typographical and technical
6/75	errors.
B	This revision corrects various typographical and technical
12/75	errors, Changes have been made to the block transfer instruc-
	tions. The title of this manual has been changed to the
	Preliminary CAL Assembler Reference Manual.

Publication Number 2240000

Written by:
Paul M. Johnson
CRAY RESEARCH, INC.
7850 Metro Parkway
Suite 213
Minneapolis, MN. 55420

CONTENTS

Introduction 1

Summary of machine characteristics 1

Principal operating registers 1

- A registers 1
- B registers 1
- S registers 2
- T registers 2
- V registers 2
- VL register 2
- VM register 2
- P register 2

Supporting registers 5

- BA register 5
- LA register 5
- XA register 5
- F register 5
- M register 6

Input/Output 6

Comparison of scalar and vector processing 6

Functional units 7

Instruction formats 7

Special register values 9

Instruction buffers. 9

Data formats 9

CAL language structure 11

Statement format 11

Location field 11
Result field 11
Operand field 11
Comments field 11

Coding conventions 12

Comments statement 12

Lower case in comments 12

Symbols 12

Special element, * 13

"P." prefix 13

"W." prefix 13

Expressions 13

CPU register designation 14

Pseudo-instructions 14

ABS 14
BSS 14
BSSZ 15
CON 15
EJECT 15
END 15
ENTRY 16
= 16
IDENT 16
LIST 16
ORG 17

CPU instructions 18

Appendix A. Summary of CPU instructions 60

Appendix B. Instruction timing 66

Appendix C. Coding examples 69

Appendix D. Use of the NOVA CAL assembler 71

Appendix E. Assembly errors 72

Appendix F. Description of binary output 73

FIGURES

Figure 1. Registers block diagram 3

Figure 2. Exchange package 4

Figure 3. Instruction formats 8

Figure 4. Data formats 10

Introduction

The Cray Research CRAY-1 computer is a powerful general purpose computer incorporating vector capabilities and a large, fast bi-polar memory. Vector processing provides result rates greatly exceeding the result rates of conventional scalar processing. The benefits of vector processing are visible even for short vectors. This manual introduces the characteristics of the CRAY-1 and describes the CRAY-1 assembly language (CAL).

Summary of machine characteristics

- 64-bit word
- 2's complement arithmetic
- scalar and vector processing modes
- 12 fully-segmented functional units
- eight 24-bit A registers
- sixty-four 24-bit B registers
- eight 64-bit S registers
- sixty-four 64-bit T registers
- eight 64-element V registers, 64 bits per element
- 4 instruction buffers of 64 parcels each
- 12.5 nanosecond clock period
- 1,048,576 words of bi-polar memory (64 bits and one parity bit) arranged in 16 banks
- 4 clock period bank cycle time
- 1 word/clock period transfer rate to B, T and V registers
- 1 word/two clock periods transfer rate to A and S registers
- 4 words/clock period transfer rate to instruction parcel buffers
- 12 full-duplex I/O channels

Principal operating registers

A registers

The eight 24-bit A registers are primarily used as address registers for memory references and as index registers. They are individually designated by the symbols A0, A1, A2, A3, A4, A5, A6 and A7. Data flows between these registers and the B, S and VL registers. Data may be directly transferred between the A registers and memory.

B registers

The sixty-four 24-bit B registers provide rapid-access temporary storage for the A registers. They are individually designated by the symbols B0, B1, B2, ..., B77. Data may be directly transferred between the B registers and memory.

S registers

The eight 64-bit S registers are the principal scalar registers for the CPU. They are individually designated by the symbols S0, S1, S2, S3, S4, S5, S6 and S7. These registers serve as source and destination registers in scalar arithmetic and logical instructions. They may also furnish one operand in vector instructions. Data flows between these registers and the A, T, V, and VM registers. Data may be directly transferred between the S registers and memory.

T registers

The sixty-four 64-bit T registers provide rapid-access temporary storage for the S registers. They are individually designated by the symbols T0, T1, T2, . . .,T77. Data may be directly transferred between the T registers and memory.

V registers

The eight 64-element V registers are the operating registers for vector computations. Each element is 64 bits. The V registers are individually designated by the symbols V0, V1, V2, V3, V4, V5, V6 and V7. These registers serve as source and destination registers in vector arithmetic and logical instructions. Data flows between these registers and the S registers. Data may be directly transferred between the V registers and memory.

VL register

The 7-bit VL register specifies the vector length. Vector computations are performed on vectors of the length specified by the contents of VL.

VM register

The 64-bit VM register contains a vector mask to control register selection in the vector merge instructions (146-147). Each bit of the VM register corresponds to a vector element.

P register

The 24-bit P register specifies the parcel address of the current program instruction. The high order 22 bits specify a memory address and the low order 2 bits specify the parcel number.

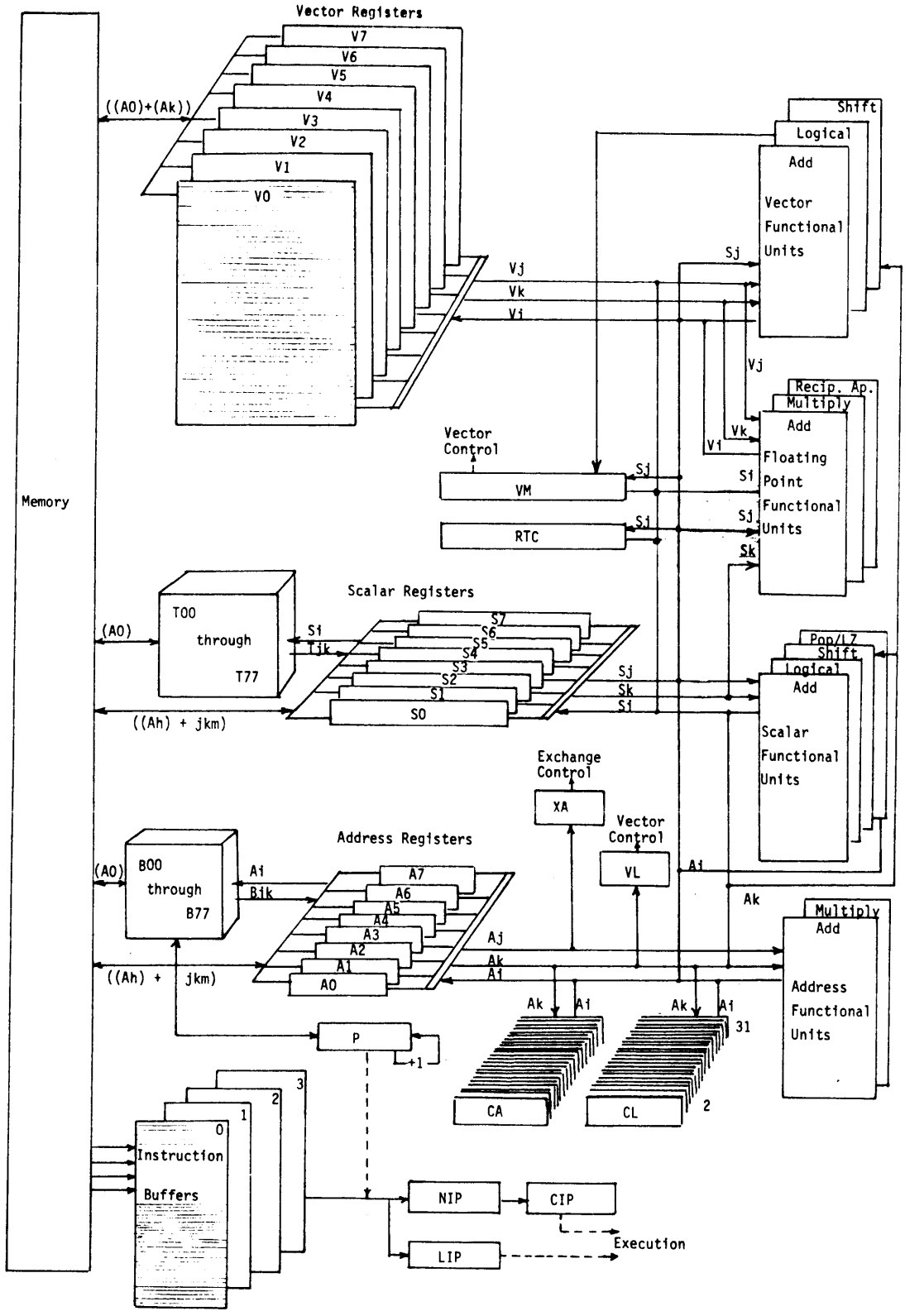
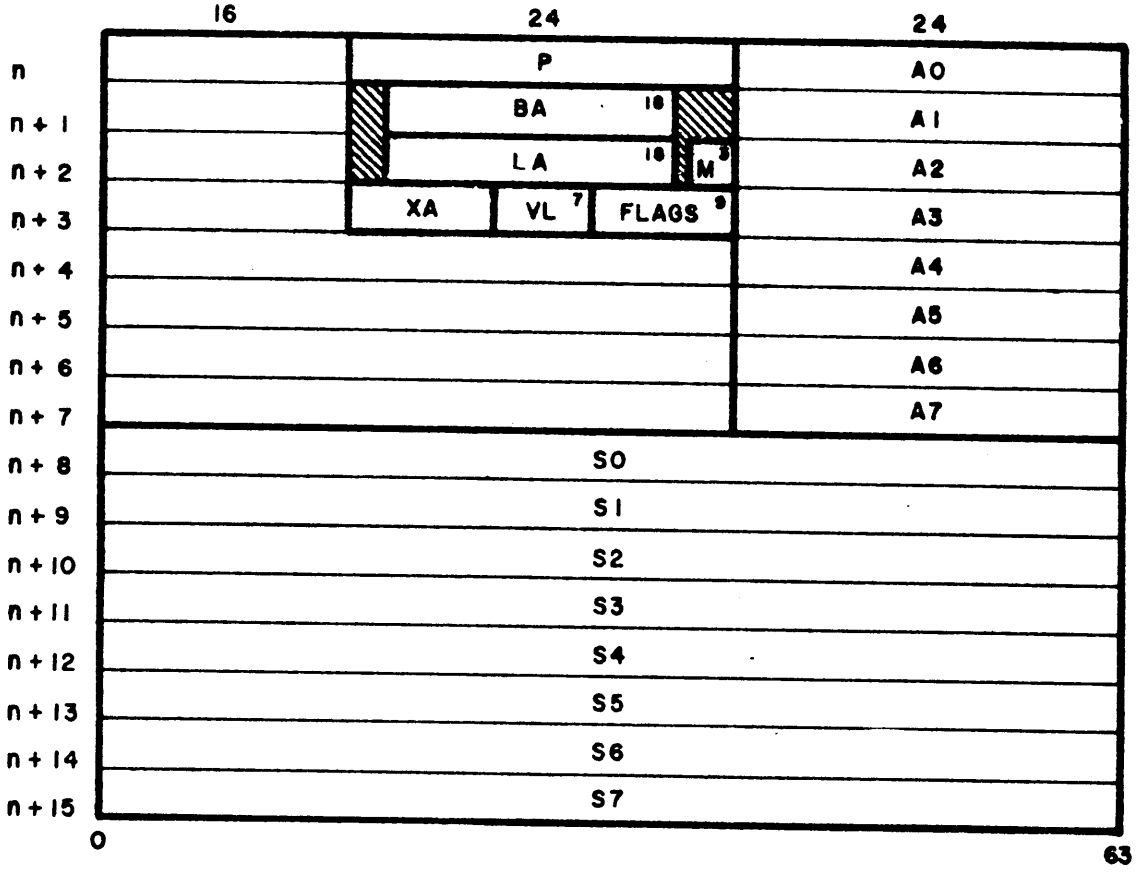


Figure 1. Registers block diagram



FLAGS

Bit*	
31	Console Interrupt
32	RTC Interrupt
33	Floating Point Error (Set on Scalar Reference Only)
34	Operand Range
35	Program Range
36	Storage Parity
37	I/O Interrupt
38	Error Exit
39	Normal Exit

MODES

Bit*	
31	Interrupt on Floating Point
32	Interrupt on Storage Parity
33	Monitor Mode

P = Program Address
BA = Base Address
LA = Limit Address
XA = Exchange Address
VL = Vector Length

* Bit position from left of word

Figure 2. Exchange package

Supporting registers

The CPU contains a number of registers which support the operating registers in the execution of programs. These registers are loaded with new information during the execution of an exchange sequence. The information is not altered during the execution interval for an exchange package. These registers are listed below with a description of the individual function performed.

BA register

This 18-bit register holds the base address during the execution interval for each exchange package. The contents of this register is interpreted as the upper 18 bits of a 22-bit memory address. The lower 4 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.

LA register

This 18-bit register holds the limit address during the execution interval for each exchange package. The contents of this register is interpreted as the upper 18 bits of a 22-bit memory address. The lower 4 bits of the address are assumed zero. The BA and LA registers together provide memory protection. No memory references may be made below BA nor at or above LA. Such a reference will cause the program or operand range flag to be set and the execution interval of the exchange package will be terminated.

XA register

This 8-bit register holds the upper eight bits of a 12-bit exchange address during the execution interval for each exchange package. The low order 4 bits of the exchange address are assumed zero.

When the execution interval terminates, the exchange operation exchanges the contents of the registers with the contents of the exchange package at (XA)*16 in memory. The exchange operation saves the contents of the A, S, P, and VL registers and the supporting registers BA, LA, XA, M and F.

F register

This 9-bit register contains flags which are set to indicate the conditions causing an exchange operation. The interrupt conditions are:

- Normal exit
- Error exit
- I/O interrupt
- Storage parity
- Program range
- Operand range
- Floating point overflow (scalar only)
- External clock interrupt
- Console interrupt

M register

This 3-bit register specifies the modes for generation of interrupts. All interrupts are inhibited when the monitor mode bit is set. Interrupts on storage parity errors are enabled when the storage parity mode bit is set. Interrupts on scalar floating point overflow are enabled when the floating point mode bit is set.

Input/Output

There are twenty-four I/O channels, of which twelve are input channels and twelve are output channels. The channels are assigned the numbers 2 through 25. The channels are divided into four groups as follows:

Group 1	Input channels 2, 6, 10, 14, 18, 22
Group 2	Output channels 3, 7, 11, 15, 19, 23
Group 3	Input channels 4, 8, 12, 16, 20, 24
Group 4	Output channels 5, 9, 13, 17, 21, 25

Each input channel consists of a data channel (16 data bits and 3 control bits), a 64-bit assembly register, a current address (CA) register, and a channel limit address (CL) register. Each input channel can cause a CPU interrupt condition when the current address equals the limit address register value or when the input device sends a disconnect.

Each output channel consists of a data channel (16 data bits and 3 control bits), a 64-bit disassembly register, a current address (CA) register, and a channel limit address (CL) register. Each output channel can cause a CPU interrupt condition when the current address equals the limit address register value. A disconnect is sent on the output channel after the last word of a record is sent and acknowledged.

Comparison of scalar and vector processing

Scalar instructions apply a function to one or two operands in registers and enter the result into a register. The addition of two integers in S1 and S2, entering the sum into S3, is an example of a scalar instruction. Vector instructions apply a function to sets of operands called vectors. Suppose one wanted to perform several additions like the one above. One could execute a small loop which would perform one addition per pass, saving S3 sums as they are generated. Alternatively, one could enter the addends into elements of one V register and the augends into elements of another V register and then execute a single vector addition instruction. The set of addends, the set of augends, and the set of sums are vectors. Vector processing provides much higher result rates than can be obtained by conventional scalar processing.

Functional units

There are twelve functional units in the computation section of the CPU. Each is a specialized unit implementing algorithms for a portion of the instructions. Each unit is independent of the other units and a number of functional units may 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. There is no information retained in a functional unit for reference in subsequent instructions. These units operate essentially in three-address mode with very limited source and destination addressing.

Three functional units provide 24-bit results to the A registers only:

- integer add
- integer multiply
- population count

Three functional units provide 64-bit results to the S registers only:

- integer add
- shift
- logical

Three functional units provide 64-bit results to the V registers only:

- integer add
- shift
- logical

Three functional units provide 64-bit results to either the S or V registers:

- floating add
- floating multiply
- reciprocal approximation

All functional units have one clock period segmentation. This means that the information arriving at the unit, or moving within the unit, is captured and held in a new set of registers at the end of every clock period. It is therefore possible to start a new set of operands for unrelated computation into a functional unit each clock period even though the unit may require more than one clock period to complete the calculation. 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. Functional units servicing the vector instructions produce one result per clock period.

Instruction formats

Figure 3 illustrates the five instruction formats for the CRAY-1. Each instruction is either a one-parcel (16-bit) instruction or a two-parcel (32-bit) instruction. Two-parcel instructions may begin in the fourth and last parcel position within a word and end in the first parcel position of the next word. The assembler lists a parcel address as a word address followed by a one-character alphabetic (a-d) parcel identifier.

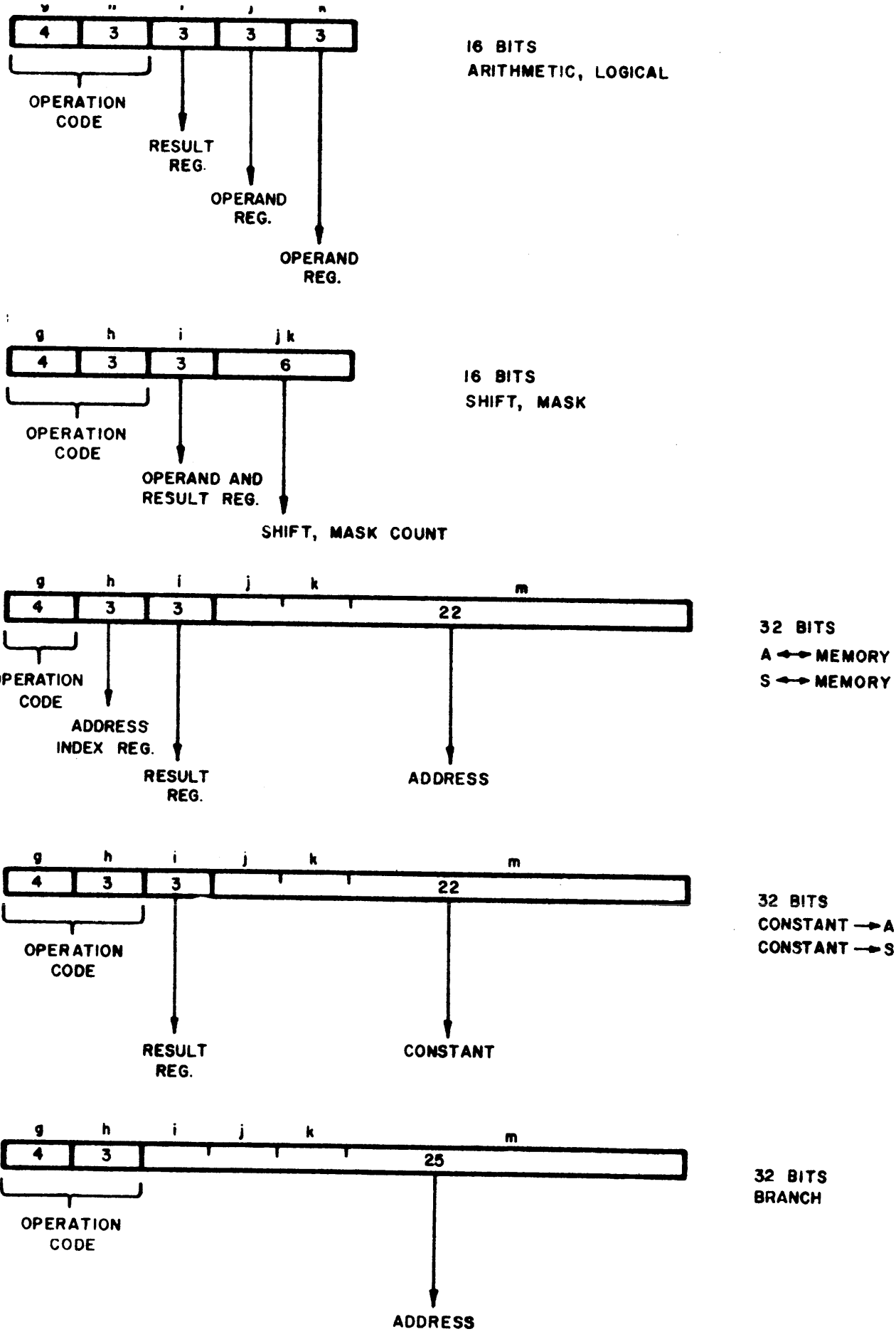


Figure 3. Instruction formats

Special register values

S0 and A0 provide special values when they are designated in the j or k portions of an instruction. In these cases the special value is used as the operand and the contents of S0 or A0 is ignored. If an S0 or A0 operand is designated in the i portion of an instruction, the actual contents of S0 or A0 is used as the operand. The instruction descriptions enumerate the uses of the special register values where they are meaningful.

register	value
Ai,i=0	A0
Aj,j=0	0
Ak,k=0	1
Si,i=0	S0
Sj,j=0	0
Sk,k=0	2^{63}

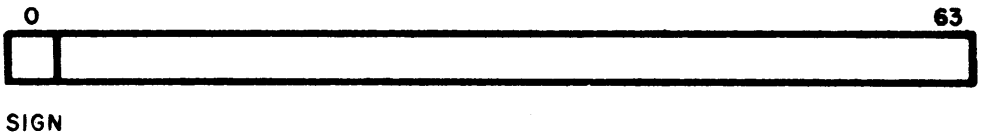
Instruction buffers

There are four instruction buffers each consisting of sixty-four 16-bit registers. All instructions are executed from the instruction buffers. An instruction buffer supplies instructions to the next instruction parcel (NIP) and the current instruction parcel (CIP) registers. Associated with each instruction buffer is a base address register that specifies the high order 18 bits of the parcel addresses contained in the instruction buffer. The base address registers are scanned each clock period. If the high order 18 bits of the P register matches one of the base addresses, the proper instruction is selected from the instruction buffer and sent to the NIP register. The instruction is moved to the CIP register for execution. The second parcel of a 2-parcel instruction resides in the NIP register when the instruction issues.

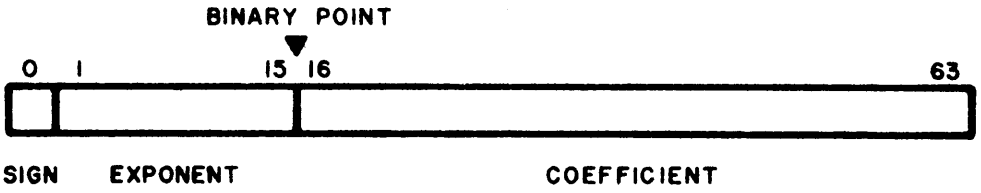
When the high order 18 bits of the P register do not match any instruction buffer base address, an "out of buffer" condition exists and instructions are read to an instruction buffer from memory. When an "out of buffer" condition occurs, the instruction buffer that receives the instruction is determined by a 2-bit counter. Each occurrence of an "out of buffer" condition causes the counter to be incremented. The first four instruction parcels in an instruction buffer are always from bank 0, however, the first parcels read into an instruction buffer always include the parcel specified by the contents of the P register.

Data formats

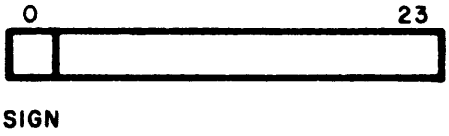
Figure 4 illustrates the data formats for integers and floating point quantities. The range for floating point quantities is $[10^{-5000}, 10^{2500}]$. Normalized floating point quantities are expressed as $z = y * 2^x$ where $y = 0$ or $\frac{1}{2} \leq y < 1$ and $-40000_8 \leq x < 20000_8$. The exponent of x is expressed in excess-40000_8 notation. The exponent of a floating point quantity is obtained by adding 40000_8 to the true exponent. Overflow is indicated by an exponent exceeding 57777_8.



2's COMPLEMENT INTEGER (64 BITS)



SIGNED MAGNITUDE FLOATING POINT (64 BITS)



2's COMPLEMENT INTEGER (24 BIT)

Figure 4. Data formats

CAL language structure

Statement format

A CAL language source program consists of a sequence of symbolic machine instructions,pseudo instructions and comment lines. Except for comment lines, each statement consists of a location field, a result field, an operand field, and a comments field. Each field is terminated by one or more blank characters. Statement format is essentially free field.

Statements are 80 column lines. When punched on cards, each card is considered a line. Information beyond column 72 is not interpreted by CAL but does appear on the assembly listing. Thus, columns 73-80 can be used for additional comments or sequencing.

Location field

The location field entry begins in column one or two of a new statement line and is terminated by a blank. If columns one and two are blank, the location field has no entry.

Result field

If the location field is blank, the result field can begin in column three. If the location field is nonblank, the result field begins with the first nonblank character following the location field and is terminated by one or more blanks. The result field is blank if there are no nonblank characters between the location field and column 35.

Operand field

The contents of the result field determines if any entry is required in the operand field. The operand field begins with the first nonblank character following the result field and is terminated by one or more blanks. It is blank if there are no nonblank characters between the result field and column 35.

Comments field

Comments are optional and begin with the first nonblank character following the operand field, or, if the operand field is missing, begin no earlier than column 35.

Coding conventions

The following coding convention should be adopted to assure uniformity of all CRAY-1 systems code:

Column	Contents
1	Asterisk (comments statement only)
1-8	Location field entry, left-justified
9	Blank
10-18	Result field entry, left-justified
19	Blank
20-34	Operand field entry, left-justified
35	Beginning of comments

Comments statement

A comments statement is designated by either an asterisk in column 1 or by blanks in columns 1-34. Comments statements are listed in assembler output but have no other effect on assembly.

Lower case in comments

Since the standard keypunch requires multipunching of lower case characters, an escape character is provided to indicate that succeeding alphabetic characters (A-Z) are to be converted to lower case. The conversion is performed only for comments statements and comments fields. Conversion is terminated by a subsequent occurrence of the escape character, which may be on a different card, or by the end of the program. When in lower case mode, a single alphabetic character may be capitalized by prefixing the capitalization character. The capitalization character has no effect if not followed by an alphabetic character.

	character	card code	ASCII code
Escape character	SUB	7-8-9	032
Capitalization character	EOT	7-9	004

Symbols

A symbol is a string of 1-8 characters that defines a value and its associated attributes. The first character must be alphabetic (A-Z), @ or $. Second and successive characters may also be digits (0-9) or =. A symbol may have a word address or a parcel address attribute, or neither of these. A symbol is a parcel address if it appears in the location field of an instruction. A symbol is a word address if it appears in the location field of a CON, BSS, or BSSZ pseudo instruction. The "=" statement can be used to define a symbol with either attribute.

Special element, *

The use of the special element * in an expression causes the assembler to replace it with the current value of the location counter.

"P." prefix

A symbol or constant may be prefixed by a "P." to specify the attribute of parcel address. If a symbol, sym, has the attribute of word address, the value of P.sym is the value of sym multiplied by four. A "P." prefix to a symbol with neither word nor parcel address attributes or to a constant does not cause the value to be multiplied by four, but it can be used to assign the parcel address attribute to a symbol being defined by an "=" statement.

"W." prefix

A symbol or constant may be prefixed by "W." to specify the attribute of word address. If a symbol, sym, has the attribute of parcel address, the value of W.sym is the value of sym divided by four. A "W." prefix to a symbol with neither word nor parcel address attributes or to a constant does not cause the value to be divided by four, but it can be used to assign the word address attribute to a symbol being defined by an "=" statement.

Expressions

Expressions are evaluated from left to right without regard for operator (+, -, * and /) precedence. Expressions in branch instructions (006-017) must not evaluate to type "word address". Expression elements may be one of the following forms:

```
*
symbol
octal constant
O'nnnn    (nnnn, an octal constant)
D'nnnn    (nnnn, a decimal constant)
A'cccc'   (cccc, a character string)
'cccc'
```

A'cccc' or 'cccc' left-justifies the character string in a 64-bit field with blank fill. A suffix may be used to specify an alternate justification or fill: H - same as no suffix, L - left-justified, zero fill, or R - right-justified, zero fill. An apostrophe in a character string is represented as two apostrophes. A null expression is given the value zero.

CPU register designation

A, S, and V registers are designated by suffixing a single octal digit, n (An, Sn, Vn). B and T registers are designated by suffixing one or two octal digits (Bnn, Tnn). A symbol may be used in place of a B or T register number if the register name and number are separated by a period. The symbol must have been previously defined. For example,

```
RTNADDR  =  14
         J  B.RTNADDR
```

accomplishes the same thing as

```
         J  B14  .
```

Pseudo instructions

Three pseudo instructions are required for an assembly: IDENT, ENTRY, and END. IDENT must be the first source statement. END signals the termination of source statements for a program. Statements preceding the first IDENT or between a succeeding END and subsequent IDENT are interpreted as comments.

ABS

The ABS pseudo instruction specifies that the program is absolute. This pseudo must precede any BSS, BSSZ, CON, = or instruction. It has no real purpose for the NOVA CAL assembler and may be omitted, but it is implemented to facilitate the eventual relocatable module capability.

Example:

location	result	operand	comments
1	10	20	35
	ABS		

BSS

The BSS pseudo instruction causes a block of storage to be reserved. The location counter is first rounded to the next word boundary (force upper), and then the number of words specified by the operand field expression is reserved. Unused parcels are padded with pass instructions (S1 S1&S1). A location field symbol, if present, is assigned the value of the current word address after the force upper occurs.

Example:

location	result	operand	comments
1	10	20	35
	BSS	4	
	BSS	W.*+7/10*10	Round to 10_8-word boundary

BSSZ

The BSSZ pseudo instruction causes a block of zero storage to be reserved. The location counter is first rounded to the next word boundary (force upper), and then the number of zero words specified by the operand field expression is reserved. A location field symbol, if present, is assigned the value of the current word address after the force upper occurs.

Example:

location	result	operand	comments
1	10	20	35
	BSSZ	177	

CON

The CON pseudo instruction generates a full word of binary data. This pseudo always forces upper. A location field symbol, if present, is assigned the value of the current word address after the force upper occurs.

Example:

location	result	operand	comments
1	10	20	35
	CON	7777017	

EJECT

The EJECT pseudo instruction causes the next listing line to appear on a new page. The EJECT pseudo itself is not listed. The EJECT pseudo has no effect when the E global switch is selected.

Example:

location	result	operand	comments
1	10	20	35
	EJECT		

END

The END pseudo instruction indicates the end of the program. Subsequent cards, if any, are assumed to be part of the next program.

Example:

location	result	operand	comments
1	10	20	35
	END		

ENTRY

The ENTRY pseudo instruction specifies an entry point of the program. Only one ENTRY pseudo is permitted by the initial CAL assembler. The entry point name is specified in the operand field and must subsequently appear in the location field of an instruction or "=" pseudo instruction.

Example:

location	result	operand	comments
1	10	20	35
	ENTRY	EPTNME	

=

The "=" pseudo instruction defines the symbol in the location field as having the value and attribute indicated by the expression in the operand field. Any symbol in the expression must be previously defined. If the expression is erroneous, CAL does not define the location symbol but flags an error.

Example:

location	result	operand	comments
1	10	20	35
SYMB	=	A*B+100/4	

IDENT

The name of the program is specified in the operand field of the IDENT pseudo instruction. The name must be 1-8 characters, of which the first must be alphabetic (A-Z), @ or $. Second and successive characters may also be digits (0-9) or =. The name appears in the listing heading and in the program descriptor table (PDT) of the absolute module.

Example:

location	result	operand	comments
1	10	20	35
	IDENT	PMJ	

LIST

The LIST pseudo instruction controls the listing. If the operand field is empty the listing is suppressed until encountering another LIST pseudo with a non-empty operand field, or until the end of the program.

Example:

location	result	operand	comments
1	10	20	35
	LIST	ON	

ORG

The ORG pseudo instruction specifies the origin of the program. This pseudo must precede any BSS, BSSZ, CON, = or instruction. The origin is specified in the operand field. If omitted, an origin of zero is assumed.

Example:

location	result	operand	comments
1	10	20	35
	ORG	0'100	

- **000 ERR** **Error Exit**

This instruction is treated as an error condition and an exchange jump occurs. The contents of the instruction buffers are voided by the exchange jump. If not in monitor mode, the error exit flag in the F register is set. If in monitor mode, no bits are set in the F register.

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 jump 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 jump is advanced one count from the address of the error exit instruction. The error exit instruction is not intended for use in user program code. Its purpose is to halt execution of an incorrectly coded program which jumps into an unused area of memory or into a data area.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
000000		ERR		

- 001 CA,Aj Ak — Set the channel (Aj) current address to (Ak) and begin the I/O sequence
  - CL,Aj Ak — Set the channel (Aj) limit address to (Ak)
  - CI,Aj — Clear the channel (Aj) interrupt flag and error flag
  - XA Aj — Enter the XA register with (Aj)
  - RT Sj — Enter the real time clock register with (Sj)

This instruction performs specialized functions useful to the operating system. The instruction is treated as a pass instruction if not in monitor mode or if the i designator is 5, 6 or 7.

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 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. When the j designator is 0 or when the contents of Aj is less than 2 or greater than 25, these functions are executed as pass instructions. When the k designator is 0, CA or CL is set to 1.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
001035		CA,A3	A5	
001134		CL,A3	A4	
001210		CI,A1		

When the i designator is 3, the instruction causes the exchange address (XA) register to be set to the contents of Aj. When the j designator is zero, the XA register is cleared. A monitor program activates a user job by initializing the XA register with the address of the user job's exchange package and then executing a normal exit (004).

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
001350		XA	A5	
001420		RT	S2	
001400		RT	S0	clear RTC

- 002 VL Ak 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. Vector instructions operate on vectors whose lengths are determined by subtracting one from the contents of VL; one plus the contents of the low order six bits of the result is the vector length. The maximum vector length of 64 can be achieved by setting the contents of Ak to zero or 64 before executing this instruction. When the k designator is zero, the vector length is set to one.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
020200000100		A2	D'64	VL = 64
002002		VL	A2	
022100		A1	0	VL = 64
002001		VL	A1	

- 003 VM Sj Transmit Sj to VM

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) where an operation is performed depending on the contents of VM.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
003040		VM	S4	
003000		VM	S0	VM = 0

- 004 EX Normal exit

This instruction causes an exchange jump. The contents of the instruction buffers are voided by the exchange jump. If not in monitor mode, 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 jump 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. The value of an optional operand field expression is inserted in the lower 9 bits of the instruction. All symbols in the expression must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
004000		EX		
004027		EX	27	

- 005 J Bjk Branch to (Bjk)

This instruction sets the P register to the parcel address specified by the contents of Bjk, and execution continues at that address.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
005017		J	B17	
005017		J	B.RTNADDR	(RTNADDR=17)

- 006 J exp Branch to ijkm

This instruction sets the P register to the parcel address specified by the low order 24 bits of the ijkm field, and execution continues at that address. The high order bit of the ijkm field is ignored.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
006 00002124b		J	TAG1	
006 00001752d		J	LDY3+1	
006 00004530a		J	*+3	

- 007 R exp Return jump to ijkm

This instruction sets register B00 to the address of the following parcel. The P register is then set to the parcel address specified by the low order 24 bits of the ijkm field, and 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 instruction on the contents of B00 (005).

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
007 00001142d		R	HELP	

- 010 JAZ exp Branch to ijkm if A0=0
- 011 JAN exp Branch to ijkm if A0≠0
- 012 JAP exp Branch to ijkm if A0 positive
- 013 JAM exp Branch to ijkm if A0 negative

These instructions test the contents of A0 for the condition specified. If the condition is satisfied, the P register is set to the parcel address specified by the low order 24 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. If A0 contains zero, it is considered positive.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
010 00002245b		JAZ	TAG3+2	
011 00004520a		JAN	P.CON1	
012 00002221c		JAP	TAG2	
013 00002124b		JAM	TAG1	

- 014 JSZ exp Branch to ijkm if S0=0
- 015 JSN exp Branch to ijkm if S0≠0
- 016 JSP exp Branch to ijkm if S0 positive
- 017 JSM exp Branch to ijkm if S0 negative

These instructions test the contents of S0 for the condition specified. If the condition is satisfied, the P register is set to the parcel address specified by the low order 24 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. If S0 contains zero, it is considered positive.

Examples:

Code Generated
014 00002221c
015 00002124d
016 00004540d
017 00002367c

location	result	operand	comments
1	10	20	35
	JSZ	TAG2	
	JSN	TAG1+2	
	JSP	*+3	
	JSM	TAG4	

- 020 Ai exp Transmit jkm to Ai
  Ai #exp

This two-parcel instruction enters the 22-bit quantity of the jkm field into Ai. The quantity is treated as a 22-bit positive integer; the upper bits of Ai are cleared. The assembler generates this instruction when no # symbol precedes the expression and the value of the expression exceeds 63. If all symbols in the expression have not been previously defined, this instruction is generated when the expression value is positive even though the value may be less than 64. When the # symbol precedes the expression, the expression is first evaluated and if the value is negative, it is complemented and the complemented value is stored in the jkm field. The complement is formed by changing all 1 bits to zero and all 0 bits to one. When the expression is positive, an 021 instruction is generated (see below).

Examples:

Code Generated
020200000130
020300000021
020401777777
020500051531
020600000000

location	result	operand	comments
1	10	20	35
	A2	130	
	A3	VAL+1	
	A4	1777777	
	A5	A'SY'R	
	A6	#MINUS1	MINUS1=-1

- 021 Ai exp Transmit jkm to Ai and complement
Ai #exp

This two-parcel instruction enters the 22-bit quantity of the jkm field into Ai and complements the result. The complement is formed by changing all 1 bits to zero and all 0 bits to one. This instruction is used to enter a negative number into an A register. The assembler generates this instruction when the value of the expression is negative and no # symbol precedes the expression; the jkm field will contain the complement of the expression. When the # symbol precedes the expression, the expression is first evaluated and if the value is positive it is stored in the jkm field. When the expression is negative, an 020 instruction is generated (see above).

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
021200000010		A2	#10	
021200000007		A2	-10	

- 022 Ai exp Transmit jk to Ai

This one-parcel instruction enters the 6-bit quantity of the jk field into Ai. All symbols in the expression must be previously defined. If all symbols are not previously defined, an 020 instruction is generated even though the value may be less than 64.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
022310		A3	10	

- 023 Ai Sj Transmit Sj to Ai

This instruction enters the low order 24 bits of Sj into Ai. Ai is cleared if the j designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
023410		A4	S1	

- 024 Ai Bjk Transmit Bjk to Ai

This instruction enters the contents of Bjk into Ai. A symbolic B register number must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
024517		A5	B17	
024517		A5	B.SVNTEEN	

- 025 Bjk Ai Transmit Ai to Bjk

This instruction enters the contents of Ai into Bjk. A symbolic B register number must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
025634		B34	A6	
025634		B.THRTY4	A6	

- 026 Ai PSj Population count of Sj to Ai

This instruction counts the number of one bits in Sj and enters the result into Ai. Ai is cleared if the j designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
026720		A7	PS2	

- 027 Ai ZSj Leading zero count of Sj to Ai

This instruction counts the number of leading zeroes in Sj and enters the result into Ai. Ai is set to 64 if the j designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
027130		A1	ZS3	

- 030 Ai Aj+Ak Integer sum of Aj and Ak to Ai

This instruction forms the integer sum of Aj and Ak and enters the result into Ai. No overflow is detected. Ak is transmitted to Ai when the j designator is zero and the k designator is non-zero. One is transmitted to Ai when the j and k designators are both zero. (Aj)+1 is transmitted to Ai when the j designator is non-zero and the k designator is zero. The assembler allows alternate forms for this instruction when either j or k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
030123		A1	A2+A3	
030102		A1	A2	(special form)
030230		A2	A3+1	(special form)

- 031 Ai Aj-Ak Integer difference of Aj and Ak to Ai

This instruction forms the integer difference of Aj and Ak and enters the result into Ai. No overflow is detected. The negative of Ak is transmitted to Ai when the j designator is zero and the k designator is non-zero. -1 is transmitted to Ai when the j and k designators are both zero. (Aj)-1 is transmitted to Ai when the j designator is non-zero and the k designator is zero. The assembler allows alternate forms for this instruction when either j or k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
031456		A4	A5-A6	
031102		A1	-A2	(special form)
031450		A4	A5-1	(special form)

- 032 Ai Aj*Ak 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. Ai is cleared when the j designator is zero. Aj is transmitted to Ai when the k designator is zero and the j designator is non-zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
032712		A7	A1*A2	

- 033 Ai CI Channel number of highest priority interrupt request to Ai
Ai CA,Aj Current address of channel (Aj) to Ai
Ai CE,Aj Error flag of channel (Aj) to Ai

This instruction enters channel status information into Ai. The j and k designators and the contents of Aj define the desired information. 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 non-zero. The value of the current address (CA) register for the channel is entered into Ai when the k designator is zero. The error flag for the channel is entered into the low order bit of Ai when the k designator is one. The high order bits of Ai are cleared. The error flag is cleared only by the 0012 instruction and is privileged to the monitor.

Examples:

Code Generated

```
033100
033230
033341
```

location	result	operand	comments
1	10	20	35
	A1	CI	
	A2	CA,A3	
	A3	CE,A4	

- 034 Bjk,Ai ,A0 Read (Ai) words starting at B register jk from memory starting at (A0)

This instruction is used to read the low order 24 bits of words from memory directly into the B registers. A0 contains the address in memory of the first word. The B register which is to receive the first word is specified by jk. Subsequent words are stored in consecutive B registers. Processing of the B registers is circular. B00 is processed after B77 if the count is not exhausted. The low order seven bits of (Ai) specify the number of words read. A0 in the operand field is optional.

Examples:

Code Generated

```
034407
034407
```

location	result	operand	comments
1	10	20	35
	B7,A4	,A0	
	B7,A4	,	

- 035 ,A0 Bjk,Ai Store (Ai) words starting at B register jk to memory starting at (A0)

This instruction is used to store the B registers directly into memory. A0 contains the address in memory to receive the first word. The B register which is stored at the first address is specified by jk. Subsequent B registers are stored in consecutive words in memory. Processing of the B registers is circular. B00 is processed after B77 if the count is exhausted. The low order seven bits of (Ai) specify the number of words written. A0 in the operand field is optional.

Examples:

Code Generated

```
035522
035522
```

location	result	operand	comments
1	10	20	35
	,A0	B22,A5	
	,	B22,A5	

- 036 Tjk ,Ai ,A0 Read (Ai) words starting at T register jk from memory starting at (A0)

This instruction is used to read 64-bit words from memory directly into the T registers. A0 contains the address in memory of the first word. The T register which is to receive the first word is specified by jk. Subsequent words are stored in consecutive T registers. Processing of the T registers is circular. T00 is processed after T77 if the count is not exhausted. The low order seven bits of (Ai) specify the number of words read. A0 in the operand field is optional.

Examples:

Code Generated

```
036407
036407
```

location	result	operand	comments
1	10	20	35
	T7,A4	,A0	
	T7,A4	,	

- 037 ,A0 Tjk,Ai Store (Ai) words starting at T register jk to memory starting at (A0)

This instruction is used to store the T registers directly into memory. A0 contains the address in memory to receive the first word. The T register which is stored at the first address is specified by jk. Subsequent T registers are stored in consecutive words in memory. Processing of the T registers is circular. T00 is processed after T77 if the count is not exhausted. The low order seven bits of (Ai) specify the number of words written. A0 in the operand field is optional.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
037522		,A0	T22,A5	
037522		,	T22,A5	

- 040 Si exp Transmit jkm to Si
  Si #exp

This two-parcel instruction enters the 22-bit quantity of the jkm field into Si. The quantity is treated as a 22-bit positive integer; the upper bits of Si are cleared. The assembler generates this instruction when no # symbol precedes the expression and the value of the expression is positive. When the # symbol precedes the expression, the expression is first evaluated and if the value is negative, it is complemented and the complemented value is stored in the jkm field. The complement is formed by changing all 1 bits to zero and all 0 bits to one. When the expression is positive, an 041 instruction is generated (see below).

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
040200000130		S2	130	
040300000021		S3	VAL+1	
040401777777		S4	1777777	
040500051531		S5	A'SY'R	
040600000000		S6	#MINUS1	MINUS1=-1

- 041 Si exp Transmit jkm to Si and complement
Si #exp

This two-parcel instruction enters the 22-bit quantity of the jkm field into Si and complements the result. The complement is formed by changing all 1 bits to zero and all 0 bits to one. This instruction is used to enter a negative number into an S register. The assembler generates this instruction when the value of the expression is negative and no # symbol precedes the expression; the jkm field will contain the complement of the expression value. When the # symbol precedes the expression, the expression is first evaluated and if the value is positive, it is stored in the jkm field. When the expression is negative, an 040 instruction is generated (see above).

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
041200000000		S2	-1	
041300000002		S3	#2	
041401777776		S4	-1777777	
041400000003		S4	#VAL2	(VAL2=3)

- 042 Si <exp Form ones mask in Si from the right
Si #>exp

This instruction is used to generate a mask of ones from the right. The assembler evaluates the expression to determine the mask length. If the # symbol precedes the expression the mask length is 64 minus the expression value. All symbols in the expression must be previously defined. The assembler stores 64 minus the mask length in the jk field of the instruction. The mask length must be a positive integer not exceeding 64. If the mask length is zero, an 043 instruction is generated.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
042273		S2	<5	
042273		S2	#>73	
042366		S3	<D'10	
042400		S4	<100	
043500		S5	<0	
042677		S6	1	(special form)

- 043 Si >exp Form ones mask in Si from the left
Si #<exp

This instruction is used to generate a mask of ones from the left. The assembler evaluates the expression to determine the mask length. If the # symbol precedes the expression the mask length is 64 minus the expression value. All symbols in the expression must be previously defined. The assembler stores the mask length in the jk field of the instruction. The mask length must be a positive integer not exceeding 64. If the mask length is 64, an 042 instruction is generated.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
043205		S2	>5	
043205		S2	#<73	
043312		S3	>D'10	
042400		S4	>100	
043500		S5	>0	
043600		S6	0	

- 044 Si Sj&Sk Logical product of Sj and Sk to Si

This instruction forms the logical product (AND) of Sj and Sk and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of Sj and Sk are 1 as in the following example:

(Sj) = 1100
(Sk) = 1010
(Si) = 1000

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 non-zero and the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
044234		S2	S3&S5	
044655		S6	S5&S5	S5 to S6
044307		S3	S0&S7	clear S3
044160		S1	S6&S0	get sign of S6

- 045 Si #Sk&Sj Logical product of Sj and complement of Sk to Si

This 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 1 when the corresponding bits of Sj and the complement of Sk are 1 as in the following example:

(Sj) = 1100
(Sk) = 1010
(Si) = 0100

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.

Examples:

Code Generated

location	result	operand	comments
1	10	20	35
045271	S2	#S1&S7	
045433	S4	#S3&S3	clear S4
045506	S5	#S6&S0	clear S5
045670	S6	#S0&S7	clear sign bit

- 046 Si Sj\Sk Logical difference of Sj and Sk to Si

This instruction forms the logical difference (exclusive OR) of Sj and Sk and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of Sj and Sk are different as in the following example:

(Sj) = 1100
(Sk) = 1010
(Si) = 0110

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

Examples:

Code Generated

location	result	operand	comments
1	10	20	35
046123	S1	S2\S3	
046455	S4	S5\S5	clear S4
046506	S5	S0\S6	S6 to S5
046770	S7	S7\S0	toggle sign

- 047 Si #Sj\Sk Logical difference of Sk and Sj complement to Si

This instruction forms the logical difference (exclusive OR) of Sk and the complement of Sj and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of Sj and Sk are the same, as in the following example:

(Sj) = 1100
(Sk) = 1010
(Si) = 1001

Si is set to all ones if the j and k designators have the same non-zero value. The complement of Sk is transmitted to Si if the j designator is zero and the k designator is non-zero. The assembler allows a special form for this case, as shown in the example below. All bits except the sign bit of Sj are complemented and the result is transmitted to Si if the j designator is non-zero and the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
047345		S3	#S4\S5	
047607		S6	#S7	(special form)

- 050 Si Sj!Si&Sk Scalar merge

This instruction merges the contents of Sj with Si depending on the ones mask in Sk. The result is defined by (Sj&Sk)!(Si&#Sk) as in the following example:

(Sk) = 11110000
(Si) = 11001100
(Sj) = 10101010
(Si) = 10101100

This 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 1 if the j designator is zero and the k designator is non-zero. The sign bit of Sj replaces the sign bit of Si if the j designator is non-zero and the k designator is zero. The sign bit of Si is cleared if the j and k designators are both zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
050123		S1	S2!S1&S3	
050760		S7	S6!S7&S0	

- 051 Si Sj!Sk Logical sum of Sj and Sk to Si

This 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) = 1100
(Sk) = 1010
(Si) = 1110
```

Sj is transmitted to Si if the j and k designators have the same non-zero value. Sk is transmitted to Si if the j designator is zero and the k designator is non-zero. The assembler allows a special form for this case, as in the example below. Sj, with the sign bit set to 1, is transmitted to Si if the j designator is non-zero 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.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
051472		S4	S7!S2	
051366		S3	S6!S6	
051701		S7	S0!S1	
051701		S7	S1	(special form)

- 052 S0 Si<exp Shift Si left jk places to S0

This instruction shifts Si left jk places and enters the result into S0. The assembler evaluates the expression to determine the shift count. All symbols in the expression must be previously defined. The shift count must be a positive integer not exceeding 64. If the shift count is 64, an 053 instruction is generated. The shift is end-off with zero fill. Si is not altered.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
052305		S0	S3<5	
052012		S0	S0<D'10	
052760		S0	S7<VAL	
053200		S0	S2<100	

- 053 S0 Si>exp Shift Si right 64-jk places to S0

This instruction shifts Si right (64-jk) places and enters the result into S0. The assembler evaluates the expression to determine the shift count. All symbols in the expression must have been previously defined. The shift count must be a positive integer not exceeding 64. The assembler stores 64 minus the shift count in the jk field of the instruction. If the shift count is zero, an 052 instruction is generated. The shift is end-off with zero fill. Si is not altered.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
053373		S0	S3>5	
053066		S0	S0>D'10	
053760		S0	S7>VAL	
052100		S0	S1>0	

- 054 Si Si<exp Shift Si left jk places to Si

This instruction shifts Si left jk places and enters the result into Si. The assembler evaluates the expression to determine the shift count. All symbols in the expression must have been previously defined. The shift count must be a positive integer not exceeding 64. If the shift count is 64, an 055 instruction is generated. The shift is end-off with zero fill.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
054703		S7	S7<3	
054656		S6	S6<VAL+2	
055300		S3	S3<100	

- 055 Si Si>exp Shift Si right 64-jk places to Si

This instruction shifts Si right (64-jk) places and enters the result into Si. The assembler evaluates the expression to determine the shift count. All symbols in the expression must have been previously defined. The shift count must be a positive integer not exceeding 64. The assembler stores 64 minus the shift count in the jk field of the instruction. If the shift count is zero, an 054 instruction is generated. The shift is end-off with zero fill.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
055775		S7	S7>3	
055656		S6	S6>VAL+2	
054300		S3	S3>0	

- 056 Si Si,Sj<Ak Shift Si Sj left (Ak) places to Si

This instruction left shifts the 128-bit quantity formed by concatenating Si and Sj by the amount specified in Ak. The shift is end-off with zero fill. The high order 64 bits of the result are transmitted to Si. Si is cleared if the shift count exceeds 127. The shift is a left circular shift of Si if the shift count does not exceed 64 and the i and j designators are equal and non-zero. The instruction produces the same result as the 054 instruction if the shift count does not exceed 127 and the j designator is zero. The 128-bit quantity is shifted left one place if the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
056235		S2	S2,S3<A5	
056604		S6	S6,S0<A4	
056604		S6	S6<A4	(special form)
056774		S7	S7,S7<A4	

- 057 Si Sj,Si>Ak Shift Sj Si right (Ak) places to Si

This instruction right shifts the 128-bit quantity formed by concatenating Sj and Si by the amount specified in Ak. The shift is end-off with zero fill. The low order 64 bits of the result are transmitted to Si. Si is cleared if the shift count exceeds 127. The shift is a right circular shift of Si if the shift count does not exceed 64 and the i and j designators are equal and non-zero. The instruction produces the same result as the 055 instruction if the shift count does not exceed 127 and the j designator is zero. The 128-bit quantity is shifted right one place if the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
057235		S2	S3,S2>A5	
057604		S6	S0,S6>A4	
057604		S6	S6>A4	(special form)
057774		S7	S7,S7>A4	

- 060 Si Sj+Sk Integer sum of Sj and Sk to Si

This instruction forms the integer sum of Sj and Sk and enters the result into Si. No overflow is detected. Sk is transmitted to Si if the j designator is zero and the k designator is non-zero. The high order bit of Si is set and all other bits of Si are cleared if the j and k designators are both zero, The j and k designators will normally be non-zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
060237		S2	S3+S7	
060405		S4	S0+S5	

- 061 Si Sj-Sk Integer difference of Sj and Sk to Si

This instruction forms the integer difference of Sj and Sk and enters the result into Si. No overflow is detected. The negative of Sk is transmitted to Si if the j designator is zero and the k designator is non-zero. The assembler allows a special form for this case, as in the example below. The high order bit of Si is set and all other bits of Si are cleared when the j and k designators are both zero. The k designator will normally be non-zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
061123		S1	S2-S3	
061506		S5	-S6	(special form)

- 062 Si Sj+FSk Floating sum of Sj and Sk to Si

This instruction forms the sum of the floating point quantities in Sj and Sk and enters the normalized result into Si. The result will be normalized even if the operands are unnormalized. The floating point quantity in Sk is transmitted to Si as a normalized floating point number if the j designator is zero and the k designator is non-zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
062345		S3	S4+FS5	

- 063 Si Sj-FSk Floating difference of Sj and Sk to Si

This instruction forms the difference of the floating point quantities in Sj and Sk and enters the normalized result into Si. The result will be normalized even if the operands are unnormalized. The negative of the floating point quantity in Sk is transmitted to Si as a normalized floating point number if the j designator is zero and the k designator is non-zero. The k designator is normally non-zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
063761		S7	S6-FS1	

- 064 Si Sj*FSk Floating product of Sj and Sk to Si

This instruction forms the product of the floating point quantities in Sj and Sk and enters the result into Si. The result may not be normalized if the operands are unnormalized.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
064234		S2	S3*FS4	

- 065 Si Sj*HSk Half-precision rounded floating product of Sj and Sk to Si

This 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 24 bits of the result are cleared.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
065167		S1	S6*HS7	

- 066 Si Sj*RSk Rounded floating product of Sj and Sk to Si

This instruction forms the rounded product of the floating point quantities in Sj and Sk and enters the result into Si. The result may not be normalized if the operands are unnormalized.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
066147		S1	S4*RS7	

- 067 Si Sj*ISk Reciprocal iteration

This instruction forms 2 minus the product of the floating point quantities in Sj and Sk and enters the result into Si. This instruction occurs in the divide sequence as illustrated in the example for the 070 instruction.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
067323		S3	S2*IS3	

- 070 Si /HSj Floating reciprocal approximation of Sj to Si

This instruction forms an approximation to the reciprocal of the 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 shown in the example below.

Examples:

Code Generated

location	result	operand	comments
1	10	20	35
*	Divide S1	by S2; result	to S1
070320	S3	/HS2	
064113	S1	S1*FS3	
067223	S2	S2*IS3	
064112	S1	S1*FS2	

- 071 Si Ak Transmit Ak to Si with no sign extension
Si +Ak Transmit Ak to Si with sign extension
Si +FAk Transmit Ak to Si as unnormalized floating point number
Si 0.6 Transmit constant .75*2**48 to Si
Si 0.4 Transmit constant .5 to Si
Si 1. Transmit constant 1. to Si
Si 2. Transmit constant 2. to Si
Si 4. Transmit constant 4. to Si

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

When the j designator is 0, the 24-bit value in Ak is transmitted to Si. The value is treated as an unsigned integer and the high order bits of Si are cleared. When the k designator is zero, 1 is transmitted to Si.

When the j designator is 1, the 24-bit value in Ak is transmitted to Si. The value is treated as a signed integer and the sign bit of Ak is extended to the high order bits of Si. When the k designator is zero, 1 is transmitted to 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. When the k designator is zero, an unnormalized floating point 1 is transmitted to Si.

When the j designator is 3, the constant .75*2**48 is entered into Si. This constant is normally used to extract the integer part of a floating point quantity ("fix"), as illustrated in the example below.

When the j designator is 4, 5, 6, or 7, the normalized floating point constants .5, 1., 2. and 4., respectively, are transmitted to Si.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
071705		S7	A5	
071213		S2	+A3	
071122		S1	+FA2	
071130		S1	0.6	
071240		S2	0.4	
071350		S3	1.	
071460		S4	2.	
071570		S5	4.	
	*	"fix" a floating point number in S1		
	*	separate integer and fractional parts		
071230		S2	0.6	
062312		S3	S1+FS2	
023130		A1	S3	integer part
063332		S3	S3-FS2	f.p. integer part
063113		S1	S1-FS3	fractional part

- 072 Si RT Transmit RTC to Si

This instruction enters the 64-bit value of the real time clock into Si. The clock is incremented by one each clock period. The operating system clears the real time clock when the system is initialized.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
072700		S7	RT	

- 073 Si VM Transmit vector mask to Si

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

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
073200		S2	VM	

- 074 Si Tjk Transmit Tjk to Si

This instruction enters the contents of Tjk into Si. A symbolic T register number must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
074306		S3	T6	
074566		S5	T66	
074541		S5	T.TEMP	

- 075 Tjk Si Transmit Si to Tjk

This instruction enters the contents of Si into Tjk. A symbolic T register number must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
075306		T6	S3	
075566		T66	S5	
075541		T.TEMP	S5	

- 076 Si Vj,Ak Transmit Vj element (Ak) to Si

This instruction enters the element of Vj specified by the contents of Ak into Si. The low order 6 bits of Ak are used to determine the vector element. The second element of Vj is selected if the k designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
076456		S4	V5,A6	

- 077 Vi,Ak Sj Transmit Sj to Vi element (Ak)

This instruction enters Sj into the element of Vi specified by the contents of Ak. The low order 6 bits of Ak are used to determine the vector element. The second element of Vi receives the contents of Sj if the k designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
077167		V1,A7	S6	

- 10h Ai exp,Ah Read from (Ah)+jkm to Ai

These instructions read words from memory directly into the A registers. The low order 24 bits of the 64-bit word are entered into Ai. The contents of Ah is added to the signed integer in the jkm field of the instruction to determine the memory address if the h designator is non-zero. Only the jkm field is used as the address if the h designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
100100004520		A1	CON1,A0	
100200004520		A2	CON1,0	
101300004521		A3	CON1+1,A1	
102417777777		A4	-1,A2	
103500000001		A5	1,A3	
104600004647		A6	CON,A4	
105700004647		A7	CON,A5	
106100000001		A1	1,A6	
107200000177		A2	177,A7	

- 11h exp,Ah Ai Store Ai to (Ah)+jkm

These instructions store words from the A registers directly into memory. The high order bits in memory are cleared. The contents of Ah is added to the signed integer in the jkm field of the instruction to determine the memory address if the h designator is non-zero. Only the jkm field is used as the address if the h designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
110100004520		CON1,A0	A1	
110200004520		CON1,0	A2	
111300004521		CON1+1,A1	A3	
112417777777		-1,A2	A4	
113500000001		1,A3	A5	
114600004647		CON,A4	A6	
115700004647		CON,A5	A7	
116100000001		1,A6	A1	
117200000177		177,A7	A2	

- 12h Si exp,Ah Read from (Ah)+jkm to Si

These instructions read words from memory directly into the S registers. The contents of Ah is added to the signed integer in the jkm field of the instruction to determine the memory address if the h designator is non-zero. Only the jkm field is used as the address if the h designator is zero.

Examples:

Code Generated

```
120100004520
120200004520
121300004521
122417777777
123500000001
124600004647
125700004647
126100000001
127200000177
```

```
location  result  operand      comments
1         10      20           35
          S1      CON1,A0
          S2      CON1,0
          S3      CON1+1,A1
          S4      -1,A2
          S5      1,A3
          S6      CON,A4
          S7      CON,A5
          S1      1,A6
          S2      177,A7
```

- 13h exp,Ah Si Store Si to (Ah)+jkm

These instructions store words from the S registers directly into memory. The contents of Ah is added to the signed integer in the jkm field of the instruction to determine the memory address if the h designator is non-zero. Only the jkm field is used as the address if the h designator is zero.

Examples:

Code Generated

```
130100004520
130200004520
131300004521
132417777777
133500000001
134600004647
135700004647
136100000001
137200000177
```

```
location  result  operand      comments
1         10      20           35
          CON1,A0 S1
          CON1,0  S2
          CON1+1,A1 S3
          -1,A2   S4
          1,A3    S5
          CON,A4  S6
          CON,A5  S7
          1,A6    S1
          177,A7  S2
```

- 140 Vi Sj&Vk Logical product of Sj and Vk to Vi

This instruction forms the logical products (AND) of elements of Vk with Sj and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set to 1 when the corresponding bits of Sj and the element of Vk are 1 as in the following example:

```
            (Sj) = 1100
element of Vk    = 1010
element of Vi    = 1000
```

The elements of Vi are cleared if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

```
140123
```

location	result	operand	comments
1	10	20	35
	V1	S2&V3	

- 141 Vi Vj&Vk Logical product of Vj and Vk to Vi

This instruction forms the logical products (AND) of elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set to 1 when the corresponding bits of the elements of Vj and Vk are 1 as in the following example:

```
element of Vj = 1100
element of Vk = 1010
element of Vi = 1000
```

The i designator must not equal the j or k designator. Elements of Vj are transmitted to Vi when the j and k designators are equal.

Example:

Code Generated

```
141257
141033
```

location	result	operand	comments
1	10	20	35
	V2	V5&V7	
	V0	V3&V3	

- 142 Vi Sj!Vk Logical sum of Sj and Vk to Vi

This instruction forms the logical sums (inclusive OR) of elements of Vk with Sj and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set to 1 when one of the corresponding bits of Sj and the element of Vk is 1 as in the following example:

```
          (Sj) = 1100
element of Vk  = 1010
element of Vi  = 1110
```

Elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Examples:

Code Generated

142615

```
location | result | operand | comments
         |   10   |   20    |   35
         |   V6   | S1!V5   |
```

- 143 Vi Vi!Vk Logical sum of Vj and Vk to Vi

This instruction forms the logical sums (inclusive OR) of elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set to 1 when one of the corresponding bits of the elements of Vj and Vk are 1 as in the following example:

```
element of Vj  = 1100
element of Vk  = 1010
element of Vi  = 1110
```

The i designator must not equal the j or k designator. Elements of Vj are transmitted to Vi when the j and k designators are equal.

Example:

Code Generated

143714

```
location | result | operand | comments
         |   10   |   20    |   35
         |   V7   | V1!V4   |
```

- 144 Vi Sj⊻Vk Logical difference of Sj and Vk to Vi

This instruction forms the logical differences (exclusive OR) of elements of Vk and Sj and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element are set to 1 when the corresponding bits of Sj and the element of Vk are different as in the following example:

```
          (Sj) = 1100
element of Vk  = 1010
element of Vi  = 0110
```

Elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

```
144267
```

location	result	operand	comments
1	10	20	35
	V2	S6⊻V7	

- 145 Vi Vj⊻Vk Logical difference of Vj and Vk to Vi

This instruction forms the logical differences (exclusive OR) of elements Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set when the corresponding bits of the elements of Vj and Vk are different as in the following example:

```
element of Vj  = 1100
element of Vk  = 1010
element of Vi  = 0110
```

The i designator must not equal the j or k designator. Elements of Vi are cleared when the j and k designators are equal.

Example:

Code Generated

```
145513
```

location	result	operand	comments
1	10	20	35
	V5	V1⊻V3	

- 146 Vi Sj!Vk&VM Vector merge

This instruction transmits Sj or elements of Vk to Vi depending on the contents of the vector mask (VM) register. The number of elements involved is determined by the contents of the VL register. Bit n of VM determines whether Sj or element n of Vk is transmitted to element n of Vi. Sj is transmitted if bit n is one and element n of Vk is transmitted if bit n is zero. An element of Vi is cleared when the corresponding bit of VM is one and the j designator is zero. The i and k designators cannot be equal.

Example. Assume the following initial register conditions:

```
(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
```

After executing the instruction V7 S2!V6&VM the first four elements of V7 have been modified as follows:

```
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.

Example:

Code Generated

146314

location	result	operand	comments
1	10	20	35
	V3	S1!V4&VM	

- 147 Vi Vj!Vk&VM Vector merge

This instruction transmits elements of Vj or Vk to Vi depending on the contents of the vector mask (VM) register. The number of elements involved is determined by the contents of the VL register. Bit n of VM determines whether element n of Vj or Vk is transmitted to element n of Vi. Element n of Vj is transmitted if bit n is one and element n of Vk is transmitted if bit n is zero. The i designator must not equal the j or k designator.

Example. Assume the following initial register conditions:

```
                (VL) = 4
                (VM) = 0 60000 0000 0000 0000 0000
     element 0 of V2  = 1
     element 1 of V2  = 2
     element 2 of V2  = 3
     element 3 of V2  = 4
     element 0 of V3  = -1
     element 1 of V3  = -2
     element 2 of V3  = -3
     element 3 of V3  = -4
```

After executing the instruction V1 V2!V3&VM the first four elements of V1 have been modified as follows:

```
     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.

Example:

Code Generated

location	result	operand	comments
1	10	20	35
147567	V5	V6!V7&VM	

- 150 Vi Vj<Ak Shift Vj left (Ak) places to Vi

This instruction shifts elements of Vj left by the amount specified in Ak and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. For each element the shift is end-off with zero fill. Elements of Vi are cleared if the shift count exceeds 63. The i and j designators cannot be equal. Elements are shifted left one place if the k designator is zero.

Example:

Code Generated

```
150123
```

location	result	operand	comments
1	10	20	35
	V1	V2<A3	

- 151 Vi Vj>Ak Shift Vj right (Ak) places to Vi

This instruction shifts elements of Vj right by the amount specified in Ak and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. For each element the shift is end-off with zero fill. Elements of Vi are cleared if the shift count exceeds 63. The i and j designators cannot be equal. Elements are shifted right one place if the k designator is zero.

Example:

Code Generated

```
151341
```

location	result	operand	comments
1	10	20	35
	V3	V4>A1	

- 152 Vi Vj,Vj<Ak Double shift Vj left (Ak) places to Vi

This instruction left shifts 128-bit quantities from Vj by the amount specified in Ak and enters the result into Vi. Element n of Vj is concatenated with element n+1 and the 128-bit quantity is shifted left by the amount specified in Ak. The shift is end-off with zero fill. The high order 64 bits are transmitted to element n of Vi. The number of elements involved is determined by the contents of the VL register. The last element of Vj, as determined by VL, is concatenated with 64 bits of zeroes. The i and j designators cannot be equal. The 128-bit quantities are shifted left one place if the k designator is zero.

Example. Assume the following initial register conditions:

```
                (VL) = 4
                (A1) = 3
element 0 of V4  = 7
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
```

After executing the instruction V5 V4,V4<A1 the first four elements of V5 have been modified as follows:

```
element 0 of V5  = 73
element 1 of V5  = 54
element 2 of V5  = 67
element 3 of V5  = 70     .
```

The remaining elements of V5 are unaltered.

Example:

Code Generated

```
152562
```

location	result	operand	comments
1	10	20	35
	V5	V6,V6<A2	

- 153 Vi Vj,Vj>Ak Double shift Vj right (Ak) places to Vi

This instruction right shifts 128-bit quantities from Vj by the amount specified in Ak and enters the result into Vi. Element n-1 of Vj is concatenated with element n and the 128-bit quantity is shifted right by the amount specified in Ak. The shift is end-off with zero fill. The low order 64 bits are transmitted to element n of Vi. The number of elements involved is determined by the contents of the VL register. 64 bits of zeroes are concatenated with the first element of Vj. The i and j designators cannot be equal. The 128-bit quantities are shifted right one place if the k designator is zero.

Example. Assume the following initial register conditions:

```
              (VL) = 4
              (A6) = 3
element 0 of V2  = 17
element 1 of V2  = 0 60000 0000 0000 0000 0005
element 2 of V2  = 1 00000 0000 0000 0000 0006
element 3 of V2  = 1 60000 0000 0000 0000 0007
```

After executing the instruction V0 V2,V2>A6 the first four elements of V0 have been modified as follows:

```
element 0 of V0  = 1
element 1 of V0  = 1 66000 0000 0000 0000 0000
element 2 of V0  = 1 30000 0000 0000 0000 0000
element 3 of V0  = 1 56000 0000 0000 0000 0000
```

The remaining elements of V0 are unaltered.

Example:

Code Generated

153714

location	result	operand	comments
1	10	20	35
	V7	V1,V1>A4	

- 154 Vi Sj+Vk Integer sum of Sj and Vk to Vi

This instruction forms the integer sums of Sj and elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. No overflow is detected. Elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

154213

location	result	operand	comments
1	10	20	35
	V2	S1+V3	

- 155 Vi Vj+Vk Integer sum of Vj and Vk to Vi

This instruction forms the integer sums of elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. No overflow is detected. The i designator must not equal the j or k designator.

Example:

Code Generated

155456

location	result	operand	comments
1	10	20	35
	V4	V5+V6	

- 156 Vi Sj-Vk Integer difference of Sj and Vk to Vi

This instruction forms the integer differences of Sj and elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. No overflow is detected. The negatives of elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

156712

location	result	operand	comments
1	10	20	35
	V7	S1-V2	

- 157 Vi Vj-Vk Integer difference of Vj and Vk to Vi

This instruction forms the integer differences of elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. No overflow is detected. The i designator must not equal the j or k designator.

Example:

Code Generated

```
157345
```

```
location  result  operand  comments
          10      20       35
          V3      V4-V5    !
```

- 160 Vi Sj*FVk Floating product of Sj and Vk to Vi

This instruction forms the products of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. Elements of Vi are cleared if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

```
160627
```

```
location  result  operand  comments
          10      20       35
          V6      S2*FV7   !
```

- 161 Vi Vj*FVk Floating product of Vj and Vk to Vi

This instruction forms the products of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated

```
161123
```

```
location  result  operand  comments
          10      20       35
          V1      V2*FV3   !
```

- 162 Vi Sj*HVk Half-precision rounded floating product of Sj and Vk to Vi

This 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 result into Vi. The low order 24 bits of the result elements are cleared. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i and k designator cannot be equal.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
162456		V4	S5*HV6	

- 163 Vi Vj*HVk Half-precision rounded floating product of Vj and Vk to Vi

This instruction forms the half-precision rounded products of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The low order 24 bits of the result elements are cleared. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
163712		V7	V1*HV2	

- 164 Vi Sj*RVk Rounded floating product of Sj and Vk to Vi

This instruction forms the rounded products of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i and k designators cannot be equal.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
164314		V3	S1*RV4	

- 165 Vi Vj*RVk Rounded floating product of Vj and Vk to Vi

This instruction forms the rounded products of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated

```
165567
```

location	result	operand	comments
1	10	20	35
	V5	V6*RV7	

- 166 Vi Sj*IVk Reciprocal iteration

This instruction forms, for each element, 2 minus the product of the floating point quantity in Sj and the floating point quantity in the element of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i and k designators cannot be equal.

Example:

Code Generated

```
166123
```

location	result	operand	comments
1	10	20	35
	V1	S2*IV3	

- 167 Vi Vj*IVk Reciprocal iteration

This instruction forms, for each element pair, 2 minus the product of the floating point quantities in the elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. This instruction occurs in the divide sequence as illustrated in the example for the 174 instruction. The i designator must not equal the j or k designator.

Example:

Code Generated

```
167456
```

location	result	operand	comments
1	10	20	35
	V4	V5*IV6	

- 170 Vi Sj+FVk Floating sum of Sj and Vk to Vi

This instruction forms the sums of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. Floating point quantities in elements of Vk are transmitted to Vi as normalized floating point quantities if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

```
170712
```

location	result	operand	comments
1	10	20	35
	V7	S1+FV2	

- 171 Vi Vj+FVk Floating sum of Vj and Vk to Vi

This instruction forms the sums of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated

```
171234
```

location	result	operand	comments
1	10	20	35
	V2	V3+FV4	

- 172 Vi Sj-FVk Floating difference of Sj and Vk to Vi

This instruction forms the differences of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The negatives of floating point quantities in elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

```
172516
```

location	result	operand	comments
1	10	20	35
	V5	S1-FV6	

- 173 Vi Vj-FVk Floating difference of Vj and Vk to Vi

This instruction forms the differences of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
173712		V7	V1-FV2	

- 174 Vi /HVj Floating reciprocal approximation of Vj to Vi

This instruction forms an approximation to the reciprocals of the floating point quantities in elements of Vj and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. This instruction occurs in the divide sequence to compute the quotients of floating point quantities as shown in the example below.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
	*	Divide elements of V1 by elements of V2;		
	*	result to	V6	
174320		V3	/HV2	
161413		V4	V1*FV3	
167532		V5	V3*IV2	
161645		V6	V4*FV5	
	*	Divide S1 by elements of V2;		
	*	result to	V6	
174320		V3	/HV2	
160413		V4	S1*FV3	
167532		V5	V3*IV2	
161645		V6	V4*FV5	

- 175 VM Vj,Z Form mask on Vj as defined by k in VM
VM Vj,N
VM Vj,P
VM Vj,M

This instruction is used to create a mask in the VM register depending on the elements of Vj. Each bit of VM corresponds to an element of Vj.

If the k designator is 0, the VM bit is set when the element is zero.

If the k designator is 1, the VM bit is set when the element is non-zero.

If the k designator is 2, the VM bit is set when the element is positive. The element is considered positive when it is zero.

If the k designator is 3, the VM bit is set when the element is negative.

The number of elements tested is determined by the contents of the VL register. VM bits which correspond to untested elements of Vj are cleared.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
175050		VM	V5,Z	
175061		VM	V6,N	
175072		VM	V7,P	
175013		VM	V1,M	

- 176 Vi ,A0,Ak Read VL words to Vi from memory starting at (A0) incremented by (Ak)

This instruction is used to read 64-bit words from memory directly into elements of the V registers. The number of elements involved is determined by the contents of the VL register. A0 contains the address of the first word which is entered into the first element of Vi. Successive words are entered into consecutive elements of Vi. The signed integer in Ak is added to the address of the current word to obtain the address of the next word. 1 is added if the k designator is zero. A0 in the second operand subfield is optional. 1 may be used in the third operand subfield if the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
176201		V2	,A0,A1	
176201		V2	,,A1	
176500		V5	,,1	

- 177 ,A0,Ak Vj Store VL words from Vj to memory starting at (A0) incremented by (Ak)

This instruction is used to store the elements of V registers directly into memory. The number of elements involved is determined by the contents of the VL register. A0 contains the address in memory to receive the first element. The signed integer in Ak is added to the current address to obtain the next address. 1 is added if the k designator is zero. A0 in the second result subfield is optional. 1 may be used in the third result subfield if the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
177032		,A0,A2	V3	
177032		,,A2	V3	
177030		,,1	V3	

APPENDIX A

Summary of CPU instructions

The general form of a memory reference is *base, index, increment.* For scalar references the *increment* field does not appear. The *base* field is defined by an expression. The *index* and *increment* fields reference A registers. For a vector memory reference, the *base* field must presently be null. A block copy reference to the B or T registers is of the form *register, index, length.* *Register* is the character B or T. The *index* field references an A register and the *length* field is an expression. A consistent format for all types of instructions allows for future expansion to treat certain formats as macro instructions.

000	ERR	Error
0010	CA,Aj Ak	Set the channel (Aj) current address to (Ak) and begin the sequence
0011	CL,Aj Ak	Set the channel (Aj) limit address to (Ak) and terminate the sequence
0012	CI,Aj	Clear the channel (Aj) interrupt flag
0013	XA Aj	Enter the XA register with (Aj)
0014	RT Sj	Enter the real time clock register with (Sj)
002	VL Ak	Transmit Ak to VL
003	VM Sj	Transmit Sj to VM
004	EX	Exit
005	J Bjk	Branch to (Bjk)
006	J exp*	Branch to ijkm
007	R exp	Branch to ijkm; set B00 = P
010	JAZ exp	Branch to ijkm if A0 = 0
011	JAN exp	Branch to ijkm if A0 ≠ 0

* The symbol "exp" denotes an expression.

012	JAP exp	Branch to ijkm if A0 positive
013	JAM exp	Branch to ijkm if A0 negative
014	JSZ exp	Branch to ijkm if S0 = 0
015	JSN exp	Branch to ijkm if S0 ≠ 0
016	JSP exp	Branch to ijkm if S0 positive
017	JSM exp	Branch to ijkm if S0 negative
020**	Ai exp Ai #exp	Transmit jkm to Ai
021**		Transmit jkm (1's) complement to Ai
022***		Transmit jk to Ai
023	Ai Sj	Transmit Sj to Ai
024	Ai Bjk	Transmit Bjk to Ai
025	Bjk Ai	Transmit Ai to Bjk
026	Ai PSj	Population count of Sj to Ai
027	Ai ZSj	Leading zero count of Sj to Ai
030	Ai Aj+Ak	Integer sum of Aj and Ak to Ai
031	Ai Aj-Ak	Integer difference of Aj and Ak to Ai
032	Ai Aj*Ak	Integer product of Aj and Ak to Ai
033	Ai CI	Channel number of highest priority interrupt request to Ai (Aj = 0)
	Ai CA,Aj	Current address of channel (Aj) to Ai (Aj ≠ 0, k = 0)
	Ai CE,Aj	Error flag of channel (Aj) to Ai (Aj ≠ 0, k = 1)
034	Bjk,Ai ,A0	Read (Ai) words starting at B register jk from memory starting at (A0)
035	,A0 Bjk,Ai	Store (Ai) words starting at B register jk to memory starting at (A0)
036	Tjk,Ai ,A0	Read (Ai) words starting at T register jk from memory starting at (A0)

** 020 or 021 instruction generated depending on the value of exp.
*** 022 instruction generated if (1) all symbols in exp have been previously defined, (2) exp not preceded by # and (3) the value of exp is less than 64.

037	,A0	Tjk,Ai	Store (Ai) words starting at T register jk to memory starting at (A0)
040*	Si exp		Transmit jkm to Si
	Si #exp		
041*			Transmit jkm (1's) complement to Si
042	Si <exp		Form ones mask in Si from the right
	Si #>exp		
043	Si >exp		Form ones mask in Si from the left
	Si #<exp		
044	Si Sj&Sk		Logical product of Sj and Sk to Si
045	Si #Sk&Sj		Logical product of Sj and complement of Sk to Si
046	Si Sj\Sk		Logical difference of Sj and Sk to Si
047	Si #Sj\Sk		Logical difference of Sk and Sj complement to Si
050	Si Sj!Si&Sk		Logical product of Si and complement of Sk OR'ed with the logical product of Sj and Sk to Si
051	Si Sj!Sk		Logical sum of Sj and Sk to Si
052	S0 Si<exp		Shift Si left jk places to S0
053	S0 Si>exp		Shift Si right jk negative places to S0
054	Si Si<exp		Shift Si left jk places
055	Si Si>exp		Shift Si right jk negative places
056	Si Si,Sj<Ak		Shift Si Sj left (Ak) places to Si
057	Si Sj,Si>Ak		Shift Sj Si right (Ak) places to Si
060	Si Sj+Sk		Integer sum of Sj and Sk to Si
061	Si Sj-Sk		Integer difference of Sj and Sk to Si
062	Si Sj+FSk		Floating sum of Sj and Sk to Si
063	Si Sj-FSk		Floating difference of Sj and Sk to Si

* 040 or 041 instruction generated depending on the value of exp.

064		Si Sj*FSk	Floating product of Sj and Sk to Si
065		Si Sj*HSk	Half-precision rounded floating product of Sj and Sk to Si
066		Si Sj*RSk	Full-precision rounded floating product of Sj and Sk to Si
067		Si Sj*ISk	Two minus the floating product of Sj and Sk to Si
070		Si /HSj	Floating reciprocal approximation of Sj to Si
	j		
071	0	Si Ak	Transmit Ak to Si with no sign extension
	1	Si +Ak	Transmit Ak to Si with sign extension
	2	Si +FAk	Transmit Ak to Si as unnormalized floating point number
	3	Si 0.6	Transmit constant .75*2**48 to Si
	4	Si 0.4	Transmit constant .5 to Si
	5	Si 1.	Transmit constant 1. to Si
	6	Si 2.	Transmit constant 2. to Si
	7	Si 4.	Transmit constant 4. to Si
072		Si RT	Transmit RTC to Si
073		Si VM	Transmit vector mask to Si
074		Si Tjk	Transmit Tjk to Si
075		Tjk Si	Transmit Si to Tjk
076		Si Vj,Ak	Transmit Vj element (Ak) to Si
077		Vi,Ak Sj	Transmit Sj to Vi element (Ak)
10h		Ai exp,Ah	Read from (Ah)+jkm to Ai (A0=0)
11h		exp,Ah Ai	Store Ai to (Ah)+jkm (A0=0)
12h		Si exp,Ah	Read from (Ah)+jkm to Si (A0=0)
13h		exp,Ah Si	Store Si to (Ah)+jkm (A0=0)
140		Vi Sj&Vk	Logical product of Sj and Vk to Vi
141		Vi Vj&Vk	Logical product of Vj and Vk to Vi
142		Vi Sj!Vk	Logical sum of Sj and Vk to Vi
143		Vi Vj!Vk	Logical sum of Vj and Vk to Vi
144		Vi Sj\Vk	Logical difference of Sj and Vk to Vi
145		Vi Vj\Vk	Logical difference of Vj and Vk to Vi

146	Vi Sj!Vk&VM	Transmit Sj if VM bit = 1, transmit Vk if VM bit = 0 to Vi
147	Vi Vj!Vk&VM	Transmit Vj if VM bit = 1, transmit Vk if VM bit = 0 to Vi
150	Vi Vj<Ak	Shift Vj left (Ak) places to Vi
151	Vi Vj>Ak	Shift Vj right (Ak) places to Vi
152	Vi Vj,Vj<Ak	Double shift Vj left (Ak) places to Vi
153	Vi Vj,Vj>Ak	Double shift Vj right (Ak) places to Vi
154	Vi Sj+Vk	Integer sum of Sj and Vk to Vi
155	Vi Vj+Vk	Integer sum of Vj and Vk to Vi
156	Vi Sj-Vk	Integer difference of Sj and Vk to Vi
157	Vi Vj-Vk	Integer difference of Vj and Vk to Vi
160	Vi Sj*FVk	Floating product of Sj and Vk to Vi
161	Vi Vj*FVk	Floating product of Vj and Vk to Vi
162	Vi Sj*HVk	Half-precision rounded floating product of Sj and Vk to Vi
163	Vi Vj*HVk	Half-precision rounded floating product of Vj and Vk to Vi
164	Vi Sj*RVk	Rounded floating product of Sj and Vk to Vi
165	Vi Vj*RVk	Rounded floating product of Vj and Vk to Vi
166	Vi Sj*IVk	Two minus the floating product of Sj and Vk to Vi
167	Vi Vj*IVk	Two minus the floating product of Vj and Vk to Vi
170	Vi Sj+FVk	Floating sum of Sj and Vk to Vi
171	Vi Vj+FVk	Floating sum of Vj and Vk to Vi
172	Vi Sj-FVk	Floating difference of Sj and Vk to Vi
173	Vi Vj-FVk	Floating difference of Vj and Vk to Vi
174	Vi /HVj	Floating reciprocal approximation of Vj to Vi

	k		
175	0	VM Vj,Z	VM = 1 where Vj = 0
	1	VM Vj,N	VM = 1 where Vj ≠ 0
	2	VM Vj,P	VM = 1 where Vj positive
	3	VM Vj,M	VM = 1 where Vj negative
176		Vi ,A0,Ak	Read VL words to Vi from memory starting at (A0) incremented by (Ak)
177		,A0,Ak Vj	Store VL words from Vj to memory starting at (A0) incremented by (Ak)

Special forms recognized by the assembler

Ai	Ak	Transmit Ak to Ai	
Ai	Aj+1	Transmit Aj+1 to Ai	
Ai	Aj-1	Transmit Aj-1 to Ai	
Ai	-Ak	Negative of Ak to Ai	
B,Ai,exp	,	B,Ai,exp	,A0
,	B,Ai,exp	,A0	B,Ai,exp
T,Ai,exp	,	T,Ai,exp	,A0
,	T,Ai,exp	,A0	T,Ai,exp
Si	<100	Full word of ones to Si	
Si	>100		
Si	<0	Transmit zero to Si	
Si	>0		
Si	0		
Si	1	Transmit one to Si	
Si	#Sk	Complement of Sk to Si	
Si	Sk	Transmit Sk to Si	
Si	Si<Ak	Shift Si left (Ak) places	
Si	Si>Ak	Shift Si right (Ak) places	
Si	-Sk	Negative of Sk to Si	
Ai	exp,0	Ai	exp,A0
exp,0	Ai	exp,A0	Ai
Si	exp,0	Si	exp,A0
exp,0	Si	exp,A0	Si
Vi	,,Ak	Vi	,A0,Ak
Vi	,,1	Vi	,A0,A0
,,Ak	Vi	,A0,Ak	Vi
,,1	Vi	,A0,A0	Vi
S0	Si>0	S0	Si<0
S0	Si<100	S0	Si>100
Si	Si>0	Si	Si<0
Si	Si<100	Si	Si>100

APPENDIX B

Instruction timing

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 registers must be free.
4. The result register group input path must be free at execution time - 1. 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.

Execution times in clock periods are given below. An asterisk indicates that issue may be delayed because of a functional unit reservation by a vector instruction. Memory may be considered a functional unit for timing considerations.

(A=A register, M=memory, B=B register, S=S register, I=Immediate, C=Channel)

24-bit results:

A := M	10*	A := C	5
M := A	1*	A := A+A	2
A := B	1	A := A*A	6
B := A	1	A := pop(S)	3
A := S	1	A := lzc(S)	4
A := I	1	VL := A	1

64-bit results:

S := M	10*	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	1
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	1

Vector instructions

Four conditions must be satisfied for issue of a vector instruction:

1. The functional unit must be free.
2. The result register must be free.
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+2 clock periods except for two special cases:
   - Memory is reserved for VL+4 clock periods.
   - A shared functional unit is reserved for VL+4 clock periods if a subsequent scalar instruction requires the unit.
2. The result register is reserved for the functional unit time +(VL+2) clock periods. The result register is reserved for the functional unit time +7 clock periods if the vector length is less than 5. At functional unit time +2 (called "chain slot time") a subsequent instruction, which uses the reserved result register as an operand register and 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+1 clock periods. Vector operand registers are reserved for 6 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+5 clock periods. Scalar operand registers are not reserved.

Vector instructions produce one result per clock period. The functional unit times are given below.

functional unit	time (c.p.)
logical	2
shift	4
integer add	3
floating add	6
floating multiply	7
reciprocal approx.	14
memory	6

Memory must be quiet before issue of the B and T register block copy instructions (034-037). Subsequent instructions may not issue for 13+jk clock periods when reading data to the B and T registers (034,036). They cannot issue for 5+jk clock periods when storing data (035,037).

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 13 clock periods.

No instruction can issue in the clock period following issue of a 2-parcel instruction.

APPENDIX C

Coding examples

1. Long vectors. When vectors have more than 64 elements it is necessary to segment the vector into groups of 64 elements and a residue before processing. The following example shows an efficient way to do this.

```
            A1      FWA          vector first word address
            A2      LWA+1        vector last word address + 1
            A0      A1-A2        - vector length
            A3      A1-A2
            S2      <6
            S1      A3
            JAP     ERROR        error if vector length ≤ 0
            S1      #S1&S2
            A3      S1
            A3      A3+1         first segment length
LOOP        VL      A3           set vector length
                                 read vector segment
                                 and perform vector computations
               ...
                                 store result

            A1      A1+A3        increment current position
            A0      A1-A2
            A3      D'64
            JAN     LOOP         loop for all segments
```

2. Loop counter. An efficient way to count the number of passes through loops when the number of passes does not exceed 64.

```
            S0      >COUNT       (mask with length = loop count)
LOOP        S0      S0<1         shift mask
                                 perform computations
               ...

            JSM     LOOP         loop required number of times
```

3. Alternate tests on the contents of S registers. Usually S0 is used to test the contents of S registers for zero, non-zero, positivity or negativity. The population count and leading zero count instructions may be used to test the contents of S registers for these conditions in A0. This may be useful when S0 cannot be destroyed or when one S register test needs to be made right after another.

```
A0      PS3
JAZ     SZR         if S3 = 0

A0      PS3
JAN     SNZ         if S3 ≠ 0

A0      ZS3
JAN     SPL         if S3 ≥ 0

A0      ZS3
JAZ     SMI         if S3 < 0
```

4. Circular shifts. The double shift instructions (056 and 057) may be used to shift an S register circularly.

```
        S7      S7,S7<A2
or:     S7      S7,S7>A2
```

APPENDIX D

Use of the NOVA CAL assembler

Name: CAL

Format: CAL filename

Purpose: To assemble a CAL assembly language source file. Output may be an absolute binary file, a listing file, or both.

Switches:

Global: By default, output of an assembly is an absolute binary file (no listing file). Switches other than those specified are ignored.

/E - list only lines with errors on listing file; no effect if L or P switches not selected
/L - listing file is produced on filename.LS
/N - no absolute binary file is produced
/O - override effect of LIST pseudo-instructions; no effect if L or P switches not selected
/P - listing on printer; overridden by L switch
/X - produce cross referencing of symbol table; no effect if L or P switches not selected

Local: None

Extensions: On input, search for filename.

On output, produce filename.SV for absolute binary and filename.LS for listing (global L switch selected).

The source file name specified on the call cannot have an extension and is limited to ten characters.

Examples: CAL Z)

causes assembly of CAL source file Z, producing an absolute binary file called Z.SV.

CAL/N/L A)

causes assembly of file A, producing as output a listing file A.LS. No binary file is produced.

CAL/P/X EXAMP)

causes assembly of file EXAMP, producing an assembly listing with cross-referenced symbol table, output to the line printer, and an absolute binary file EXAMP.SV.

APPENDIX E

Assembly errors

Error Type	Definition
O	OPERAND FIELD ERROR Indicates any of a number of possible errors in the operand field. For example: - symbol or name greater than 8 characters - expression does not have proper attribute - data error; 8 or 9 encountered in octal data - syntax error
L	LOCATION FIELD ERROR Symbol in location field is erroneous.
D	DOUBLY DEFINED SYMBOL Symbol previously defined; the first definition holds.
U	UNDEFINED SYMBOL Reference to a symbol that is not defined.
R	RESULT FIELD ERROR Indicates any of a number of possible errors in the result field. For example: - symbol or name greater than 8 characters - expression does not have proper attribute - data error; 8 or 9 encountered in octal data - syntax error - ABS or ORG following instructions or = - location field symbol begins beyond column 2

APPENDIX F

Description of binary output

The absolute binary output consists of a program descriptor table (PDT) followed by a single text table (TXT) containing the absolute code.

PDT Format:

Word 0	Bits	00-03*	Table code (17)
		04-27	Word count (7)
		28-41	Number of external names (0)
		42-55	Number of entry names * 2 (2)
		56-63	Number of blocks referenced * 2 (2) (absolute block only)
Word 1	Bits	00-63	Program name (left-justified, zero fill)
Word 2	Bits	40-63	Program length
Word 3	Bits	00-63	Enter point name (left-justified, zero fill)
Word 4	Bits	00-63	Entry value
Word 5	Bits	00-63	Date (DD/MM/YY)
Word 6	Bits	00-63	Time (HH:MM:SS)

TXT Format:

Word 0	Bits	00-03	Table code (16)
		04-27	Word count (program length + 1)
		40-63	Load address

Words 1 through "program length" contain the absolute code.

*Bit positions are numbered in decimal; the high order bit is position 0.

UPDATES

If you want to receive changes and updates to this manual, fill out the following update request form and return it to the address shown on page ii of this manual. If at a later date you no longer need or want this distribution, please fill out and return the stop notice form.

---

cut here

STOP NOTICE

Please remove my name from your distribution list for the following manual.

PUBLICATION NUMBER _____

MY NAME _____ DATE _____

ORGANIZATION or FIRM _____

ADDRESS _____

_____

cut here

---

cut here

UPDATE REQUEST FORM

Please add my name to your distribution list for the following manual.

PUBLICATION NUMBER _____

MY NAME _____ DATE _____

ORGANIZATION or FIRM _____

ADDRESS _____

_____

COMMENT SHEET

Please feel free to tell us your comments, criticisms, or compliments about this publication or this product. We value your feedback. Fill in the following information, and add your comments.

MANUAL:

Title -

Revision -

EQUIPMENT:

Name -

CRI serial code -

COMMENTS:

Thank You.

Mail to: Publications

CRAY RESEARCH, INC.
7850 Metro Parkway
Suite 213
Minneapolis, MN. 55420

DEC 3 1975

Cray Research 819 Disk Controller (expected date Spring 1976; may be changes then.) Not all true for interim controller

The 819 disk controller is a Cray Research product that is packaged in a small freon cooled cabinet. It must be located close to the CRAY-1 computer mainframe and operates synchronously with it. The controller may have from one to four Control Data 819 disk storage units attached to it. Any one of these disk storage units may be transferring data through the controller at any one time. The controller may be connected to a 16 bit mini computer station in addition to the CRAY-1 computer mainframe. If this additional connection is made the station and mainframe may share the controller operation on a function by function basis.

Each of the Control Data 819 disk storage units has two ports for controllers. A second independent data path may therefore exist to each disk storage unit through another Cray Research controller or through a Control Data controller.

The Cray Research 819 disk controller is designed to position and monitor the four disk storage units simultaneously. Separate counters exist in the controller for each of the four disk storage units. The controller may therefore monitor the angular position of each disk and read or write data on any unit without waiting for unit selection and resynchronizing of a common angular position counter. The controller contains two data buffers each of one disk sector capacity. These buffers are used in both reading and writing of data for the disk storage units. The buffers are used alternately as data streams to or from the disk so that data may be read or recorded continuously in every sector of a disk cylinder without losing a disk revolution time. An error correction code is recorded in each sector of disk data so that data recorded over a flaw in the disk surface may be recovered and a flaw mark recorded in a disk reservation table without loss of data at the time of discovery.

The CRAY-1 computer mainframe communicates with the disk controller over one input channel and one output channel. These channels are specially adapted for high speed data transfer in that the communication path is synchronous and the data is organized into units of 64 bit length. The channel data paths are 16 bit parallel paths and data flows in bursts of four 16 bit section . Output data from the CRAY-1 computer mainframe to the 819 disk controller may have one of two data formats.

1) Disk controller function message
This message consists of a 64 bit computer word in which the upper three 16 bit sections are ignored by the controller. The lowest order 16 bit section of the word is translated by the controller for function specification. The message is transmitted as a one word output block from the mainframe and is terminated by a disconnect signal to the controller.

2) Disk write data block
This output data format consists of 512 computer words of 64 bits each. The data is recorded in one sector of a disk storage unit along with control and verification data which is generated by the controller in the recording process. The 512 words of data in the disk write data block do not need to contain any control information for the disk storage unit. The 512 word output block from the mainframe is terminated by a disconnect signal to the controller.

Input data from the 819 disk controller to the CRAY-1 computer mainframe may have one of two functions.

1) Disk controller response message
This message consists of one to four 64 bit computer words containing status information for evaluation by the CRAY-1 monitor program. This message is in response to a disk controller function message from the CRAY-1 mainframe and completes the communication path for monitor control of the disk storage unit operation. This message is terminated by a disconnect signal from the 819 disk controller to the CRAY-1 mainframe.

2) Disk read data block

This input format consists of 512 computer words of 64 bits each. The data is read from one sector of a disk storage unit along with the control and verification data which was recorded with it. The control and verification data is not transmitted from the disk controller to the CRAY-1 mainframe. The 512 words of data are transmitted in a block and normally the block is terminated by a disconnect signal from the disk controller to the mainframe. If an abnormal condition is detected by the controller in the reading process an additional 16 bit word section is transmitted in place of the disconnect signal. This additional word section contains no data and is not intended to be read by the CRAY-1 computer program. It is intended to cause an error interrupt in the computer by overflowing a 512 word buffer boundary.

Disk Controller Function Messages

0000 xuu hhh hss sss Begin disk read
0001 xuu hhh hss sss Begin disk write
0010 xuu xxx xxx xxx Unit reservation
0011 xuu xxx xxx xxx Unit release
0100 xuu xxx xxx xxx Clear fault and return to zero
0101 xuu ccc ccc ccc Cylinder selection
0110 xuu xmm mmm mmm Margin selection
0111 xuu xxx fff fff Status readout

0000 xuu hhh hss sss Begin Disk Read

This function message causes the 819 disk controller to begin reading data from disk storage unit uu at the cylinder last selected for that unit. The first sector of data is read from head group hhhh and at sector ssss. The bit in the function code designated by x is not used.

This function message is transmitted from the requesting computer to the disk controller to begin the reading process. The proper cylinder in the disk storage unit must have been previously selected. The disk controller responds to this function message by transmitting a one word response message to the requesting computer. This response message contains 16 bits of data as follows.

1) Error flag
2) Unit number
9) Cylinder number
4) Head group number

In the case of a CRAY-1 computer request the 16 bits of response data are right justified in the 64 bit computer word. If the requested disk storage unit is not on cylinder at the time of the function request the response message is delayed until the on cylinder condition is met and verification data can be read from the disk recording. The error flag bit in the response message is set if there is a fault flag set in the

disk controller for the requested disk storage unit. The fault flags are those defined for the status readout function. The unit number, cylinder number, and head group number in the response message are the values read from the disk recording for the cylinder currently under the read/record heads for the requested disk storage unit. The computer monitor program can verify the values in the response message against the values requested.

After transmitting the response message to the requesting computer the disk controller begins monitoring the sector number for the requested disk storage unit until the requested sector is positioned for reading. That sector data is then read from the proper head group into disk controller buffer A. Transmission of the data in buffer A to the requesting computer begins as soon as the buffer has been filled and the computer has requested input data. The rate of this data transfer is controlled by the requesting computer. Data is sent by the disk controller as appropriate for each input data request. A disconnect is normally sent by the disk controller when 512 words of 64 bit length have been transmitted and another input data request is received. This terminates the block of data and empties buffer A in the disk controller. An abnormal termination of the data block occurs if a check sum error is detected in the data in buffer A or if a fault condition occurred during the reading process. In this case an additional 16 bit word section is transmitted instead of the disconnect signal.

The disk controller advances the requested sector number by one count when buffer A has been loaded with the first sector of data. If the sector number advance overflows the sector modulus (18) the sector number is returned to zero and the head group designator is advanced by one count. The disk controller begins reading data from this newly designated sector into buffer B as soon as the sector is in position and the requesting computer has requested the first word of data from buffer A. If the computer does not request the first word of data from

buffer A in time a disk revolution is lost and the controller waits for the sector to come into position on the next disk revolution.

The disk controller continues to read data from consecutive sectors as long as the computer requests input data and no error occurs in the disk controller. Data alternates between buffers A and B. If an error occurs and an abnormal transmission termination results the disk controller stops reading data and holds the error data for interrogation by the requesting computer. If the sector/head group designators are advanced beyond the head group modulus (10) the disk controller continues by reading the first sector in the disk cylinder. The disk controller remains in this reading mode until a new function message is sent to the controller by a requesting computer.

0001 xuu hhh hss sss Begin Disk Write

This function message causes the 819 disk controller to begin writing data onto disk storage unit uu at the cylinder last selected for that unit. The first sector of data is written on head group hhhh and at sector ssss. The bit in the function code designated by x is not used.

This function message is transmitted from the requesting computer to the disk controller to begin the writing process. The proper cylinder in the disk storage unit must have been previously selected. The disk controller responds by transmitting a one word response message to the requesting computer. This response message contains 16 bits of data as follows.

1) Error flag
2) Unit number
9) Cylinder number
4) Head group number

In the case of a CRAY-1 computer request the 16 bits of response data are right justified in the 64 bit computer word. If the requested disk storage unit is not on cylinder at the time of the function request the response message is delayed until the on cylinder condition is met and verification data can be read from the disk recording. The error flag bit in the response message is set if there is a fault flag set in the disk controller for the requested disk storage unit. The fault flags are those defined for the status readout function. The unit number, cylinder number, and head group number in the response message are the values read from the disk recording for the cylinder currently under the read/record heads for the requested disk storage unit. The computer monitor program can verify the values in the response message against the values requested.

After transmitting the response message to the requesting computer the disk controller waits for the first block of data. The controller remains in a write mode until data is received or until another function message is received. The requesting computer transmits a disk write data block of 512 computer words of 64 bits each. This data is loaded into the disk controller buffer A and the transmission rate is controlled by the computer. Error correction codes are generated in the disk controller as the data is loaded in buffer A and these codes are entered in the buffer along with the data at the end of the data block. The write data block is terminated by a disconnect signal from the transmitting computer.

When buffer A in the disk controller has been loaded with the data and the generated error correction words the disk controller begins monitoring the sector number for the requested disk storage unit for the proper angular position. When the requested sector is positioned properly for writing the disk controller streams the data from buffer A to the disk storage unit and records the 512 words of data and the error correction words. When this recording is complete the disk controller advances the requested sector number by one count. If the sector number advance overflows the sector modulus (18) the sector number is returned to zero

and the head group designator is advanced by one count.

The requesting computer may transmit the next block of data as soon as disk controller begins streaming the data from buffer A to the disk storage unit. The new block of data is entered in buffer B. The rate of the transmission is controlled by the computer after the disk controller accepts the first word. Error correction codes are generated in the disk controller and added at the end of the data block in the same manner that was described for buffer A.

When buffer B has been loaded with the data and the generated error correction words the disk controller monitors the sector number for the disk storage unit until the angular position is proper for recording the data in the newly designated sector. The data then streams from buffer B to the disk storage unit and is recorded in the sector following the buffer A data. If the computer is late in filling buffer B with data the disk controller will miss the proper time slot for consecutive sector recording and a disk revolution will be lost. The buffer B data will then be recorded on the next disk revolution.

The disk controller continues to read data from the computer and advance the sector designator for each data block as long as the computer sends data. The disk write data blocks alternate between buffer A and buffer B. If the sector/head group designators are advanced beyond the head group modulus (10) the disk controller continues recording data beginning at the first sector in the disk cylinder. The disk controller remains in a write mode after the last block of data is recorded until a new function message is sent by a requesting computer.

0010 xuu xxx xxx xxx Unit Reservation

This function message causes the 819 disk controller to request the reservation of disk storage unit uu. The bits designated by x in the function message are not used. If the requested disk storage unit is not busy with the other controller port the reservation is made in the unit. The unit then remains reserved until it is released by this controller. If the unit is busy with the other controller port at the time of this function message the reservation cannot be made and an error flag is set in the disk controller for this disk storage unit. This unit reservation error flag remains set until a subsequent 0010 function message when the test is made again.

0011 xuu xxx xxx xxx Unit Release

This function message causes the 819 disk controller to release the disk storage unit uu reservation. The bits designated by x in the function message are not used. The unit reservation error flag for this disk storage unit is set in the disk controller indicating that the unit is not available to this controller.

0100 xuu xxx xxx xxx Clear Fault and Return to Zero

This function message causes the 819 disk controller to clear the fault indicators in disk storage unit uu and position the read/record heads to cylinder zero. The bits designated by x in the function message are not used. If the unit reservation error flag is set in the disk controller for this unit then this function message will have no effect.

0101 xuu ccc ccc ccc Cylinder Selection

This function message causes the 819 disk controller to position the read/record heads to cylinder ccc ccc ccc in disk storage unit uu. The bit designated by x in the function message is not used. If the unit reservation error flag is set in the disk controller for this unit then this function message will have no effect.

0110 xuu xmm mmm mmm Margin Selection

This function message causes the 819 disk controller to select a margin condition in disk storage unit uu. The bits designated by x in the function message are not used. If the unit reservation error flag is set in the disk controller for this unit then this function message will have no effect. The significance of the bits designated by m in the function message are as follows.

bit 7) Set read late clock .
bit 6) Set read early clock
bit 5) Set servo offset direction
bit 4) Servo offset value 2^4
bit 3) Servo offset value 2^3
bit 2) Servo offset value 2^2
bit 1) Servo offset value 2^1
bit 0) Servo offset value 2^0

Bits 6 or 7 (but not both) in the function message may be used to recover marginal data by shifting the read strobe time early or late in the disk storage unit. This selection is effective for the next read operation only. If bits 6 and 7 are both set an error flag will be set in the disk storage unit.

Bits 0 thru 5 are used to offset the read/record heads from the normal cylinder position. If bit 5 is set the arm moves in the forward direction. If bit 5 is clear the arm moves in the backward direction. The lower order bits in the function message select one of 32 offset positions in each direction. This selection is effective for the next read operation only.

0111 xuu xxx fff fff Status Readout

This function message causes the 819 disk controller to respond with a status message to the requesting computer. The response may be one or four computer words of 64 bit length. The response is essentially immediate and gives the current status of the requested conditions. The response indicates the status of disk storage unit uu and the bits designated by f in the function message select the status type. The bits designated by x in the function message are not used. If none of the bits in the f field in the function message are set the disk controller responds with the status of the error flags in the controller itself. The bits in the f field select various status readouts from the disk storage units as follows.

bit 5) Read error correction data
bit 4) Read disk unit interlock register data
bit 3) Read disk unit offset register data
bit 2) Read disk unit head register data
bit 1) Read disk unit cylinder register data
bit 0) Read disk unit fault register data

The readout data to the requesting computer for a function message with a blank f field is as follows.

bit 4) Read data checkword error
bit 3) Parity error in computer data
bit 2) Cell counter error at index mark
bit 1) Sector verification error
bit 0) Unit reservation error flag

The readout data to the requesting computer for a function message with bit 5 set in the f field consists of four computer words of 64 bits each. These words contain the error correction data which may be used by the computer to modify the data in the read data block which was in error.

Disk Recording Information

The disk storage units each hold 2,424,176,640 bits of useful computer data when operated by a Cray Research disk controller. The data is located on cylinders, head groups, and sectors with parameters as follows.

Number of cylinders	411
Number of head groups	10
Number of sectors	18
Words per sector	512
Bits per word	64

The disk surfaces rotate at 3600 revolutions per minute, or a period of approximately 16.6 milliseconds. Maximum cylinder selection time is 80 milliseconds. The data is recorded on the disk surface using four read/record heads operating in parallel. Average data transfer rate when streaming data continuously over a given disk cylinder is 35.4 million bits per second. The total amount of data which can be streamed in a unit in this manner is a disk cylinder capacity of 5.9 million bits. The disk cylinders each contain 10 head groups and a given read/record head scans 18 sectors in a disk revolution. The disk surface under one read/record head is divided into 18 uniform sectors of 8960 cells each. Each sector contains a verification portion, followed by an intra-sector gap and then a data portion. The verification portion of each sector is recorded in a formatting operation when the disk storage unit is placed in operation in the system. This data is not altered during normal disk use. The data portion of the sector, which follows the intra-sector gap contains the data from the computer plus the error correction code generated by the disk controller. This portion of the sector is recorded, read, and rerecorded in normal operation. The cell assignment to the various portions of the sector under each read/record head are as follows.

Pre-recorded portion of sector.

300) preamble

2) sync byte

6) data

6) postamble

58) intra sector gap

Alterable portion of sector.

300) preamble

2) sync byte

8224) data

6) postamble

58) inter sector gap

The verification data pre-recorded in the first portion of each sector contains 24 bits per sector as follows.

2) Unit number

9) Cylinder number

4) Head group number

5) Sector number

4) Parity check

There is a manual switch in the disk controller for formatting the disk storage units. This switch changes the begin write position for each sector to allow the recording of the verification data and disables the error detection mechanisms.

Error Correction Code

The error correction code is a 32 bit code generated for each recording head and associated with 8192 bits of computer data. This code is generated by a Fire code generator at the input to the A and B buffers in the disk controller. As data is received from the computer and entered in the buffer the Fire code generator accumulates a 32 bit quantity which is then added to the data in the buffer. As data is received from the disk storage unit and entered in the buffer the Fire code generator accumulates a new 32 bit quantity which results from summing the data as was done during the writing sequence plus the summing of the error correction words that were recorded with the data. The result should be a zero quantity for each of the four channels of recorded data. If any of these four quantities is not zero an error flag is set in the disk controller for that read operation. This causes the streaming of data to the computer to stop after the erroneous data block and allows the computer to read the four 32 bit quantities for error correction of the data.

This particular error correction code allows the correction of a burst of 11 errors or the detection of any two errors of 11 or less. The code is generated by a 32 bit shifting register which shifts one bit position for each new bit of data processed. The register is cleared at the beginning of the operation. Each bit processed causes the register to shift right circularly by one bit position. If the incoming bit is different from checkword bit 2^{31} then bits 2^0, 2^2, 2^{11}, 2^{21}, and 2^{23} are complemented before the shift.

CRAY-1 Channel Parity

The 16 bit output channel from the CRAY-1 computer contains four parity bits associated with the 16 data bits. These parity bits are checked by the disk controller as each word arrives from the computer. If a parity error occurs the parity error flag is set in the disk controller and this information will appear as an error flag in the next response to a read or write function.

The 16 bit input channel to the CRAY-1 computer from the disk controller contains four parity bits associated with the 16 data bits. If a parity error occurs during a CRAY-1 input buffer operation the channel control module in the CRAY-1 computer interrupts the input buffer and the monitor program then responds to a short block of input data.

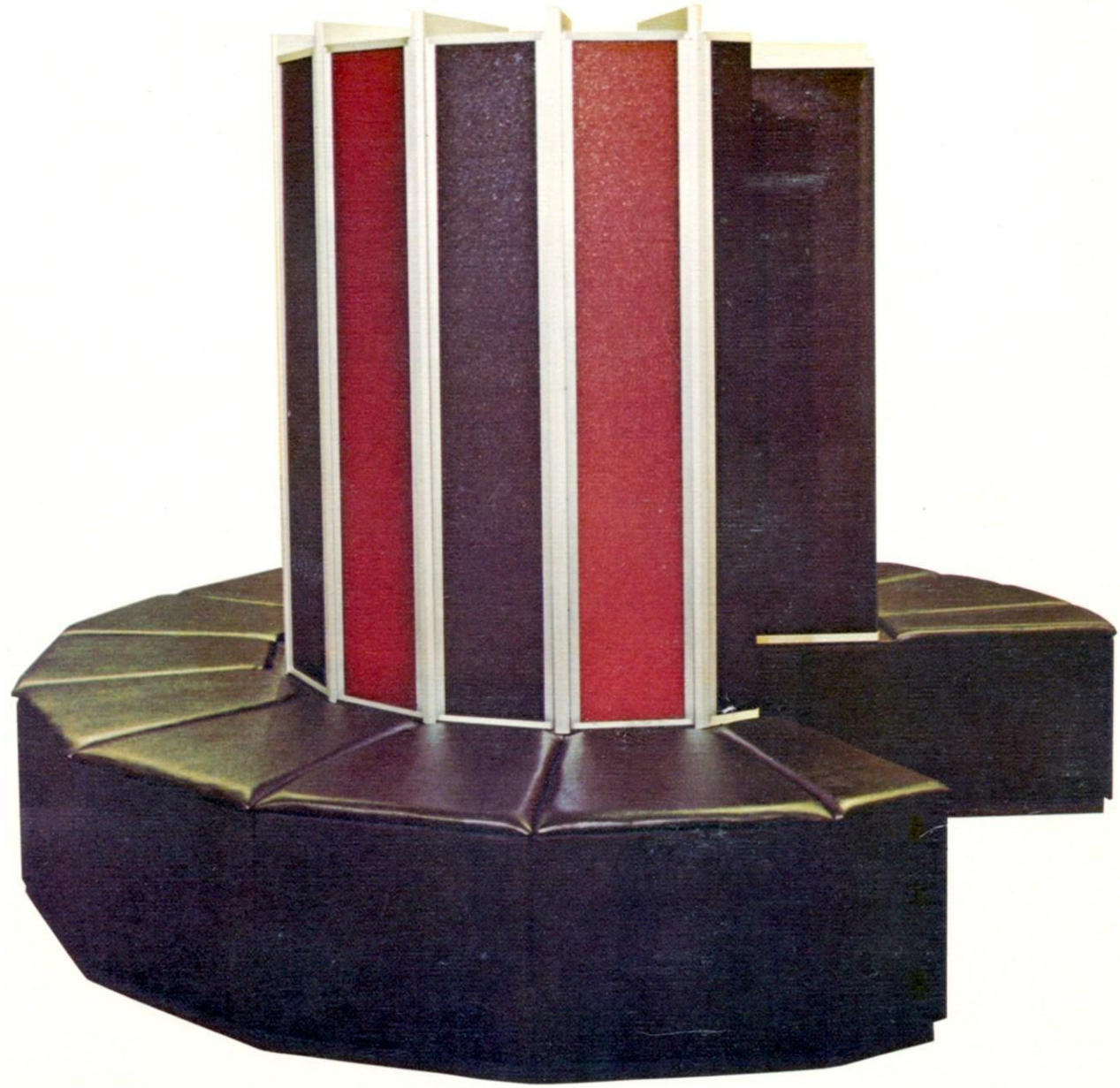
CRAY RESEARCH, INC.

CRAY-1

COMPUTER SYSTEM®

HARDWARE REFERENCE MANUAL

2240004



CRAY RESEARCH, INC.

TECHNICAL COMMUNICATIONS
7850 Metro Parkway, Suite 213, Minneapolis, MN 55420 • (612) 854-7472

PUBLICATION CHANGE NOTICE

November 4, 1977

TITLE: CRAY-1 Hardware Reference Manual

PUBLICATION NO. 2240004 REV. C

This printing obsoletes version B and applies to CRAY-1 Computer Systems starting with Serial No. 3. Revision A remains relevant for Serial 1.

CRAY-1

COMPUTER SYSTEM®

HARDWARE REFERENCE MANUAL
2240004

Copyright©1977 by CRAY RESEARCH, INC. This manual or parts thereof may not be reproduced in any form without permission of CRAY RESEARCH, INC.

RECORD OF REVISION PUBLICATION NUMBER

Revision	Print Date	Description
	1/76	Original printing
A	5/76	Reprint with revision
A-01	9/76	Corrections to pages 3-20, 3-27, 4-9, 4-10, 4-28, 4-36, 4-43, 4-55, and 4-57.
B	10/76	Reprint with revision. Addition of: Floating point range error detection Vector floating point error Error correction
B-01	2/77	Changes to exchange package (p 3-36); additions to instructions 152 and 153 (p 4-53); corrections to syndrome bit description p 5-5; corrections to instruction summary, appendix D.
B-02	7/77	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	11/77	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.

Scanned 2/2004 by gmt

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator starting with 01 for each new revision level. Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower right-hand corner. All changes are noted by a change bar along the margin of the page.

Requests for copies of CRAY RESEARCH, INC. publications should be directed to:

CRAY RESEARCH, INC.
7850 Metro Parkway
Suite 213
Bloomington, MN 55420

CONTENTS

1. INTRODUCTION 1-1

COMPUTATION SECTION 1-4
MEMORY SECTION 1-5
INPUT/OUTPUT SECTION 1-5
VECTOR PROCESSING 1-6

2. PHYSICAL ORGANIZATION 2-1

INTRODUCTION 2-1
MAINFRAME 2-1
Modules 2-1
Printed circuit board 2-4
Module assembly 2-5
Integrated circuit packages 2-5
IC high-speed logic gate 2-5
IC slow-speed logic gate 2-5
16x1 register chip 2-5
1024x1 memory chip 2-6
Resistors 2-6
Connector strips 2-6
Clock 2-7
Power supplies 2-7
PRIMARY POWER SYSTEM 2-8
COOLING 2-8
MAINTENANCE CONTROL UNIT 2-9
FRONT-END COMPUTER 2-10
EXTERNAL INTERFACE 2-10
MASS STORAGE SUBSYSTEM 2-11

3. COMPUTATION SECTION 3-1

INTRODUCTION 3-1
REGISTER CONVENTIONS 3-3
OPERATING REGISTERS 3-3
V registers 3-4
V register reservations 3-5

Vector control registers 3-6
VL register 3-6
VM register 3-6
S registers 3-7
T registers 3-8
A registers 3-8
B registers 3-9
FUNCTIONAL UNITS 3-10
Address functional units 3-11
Address add unit 3-11
Address multiply unit 3-11
Scalar functional units 3-12
Scalar add unit 3-12
Scalar shift unit 3-12
Scalar logical unit 3-13
Population/leading zero count unit 3-13
Vector functional units 3-13
Vector functional unit reservation 3-13
Recursive characteristic of vector functional units . . 3-14
Vector add unit 3-17
Vector shift unit 3-17
Vector logical unit 3-17
Floating point functional units 3-17
Floating point add unit 3-18
Floating point multiply unit 3-18
Reciprocal approximation unit 3-18
ARITHMETIC OPERATIONS 3-19
Integer arithmetic 3-19
Floating point arithmetic 3-20
Normalized floating point 3-20
Floating point range errors 3-21
Floating point add unit. 3-21
Floating point multiply unit 3-22
Floating point reciprocal approximation unit 3-22
Double precision numbers 3-23
Addition algorithm 3-23
Multiplication algorithm 3-24
Division algorithm 3-28

LOGICAL OPERATIONS 3-29
INSTRUCTION ISSUE AND CONTROL 3-30
P register 3-30
CIP register 3-31
NIP register 3-31
LIP register 3-32
Instruction buffers 3-32
EXCHANGE MECHANISM 3-35
XA register 3-35
M register 3-35
F register 3-36
Exchange package 3-36
Active exchange package 3-39
Exchange sequence 3-39
Initiated by dead start sequence 3-40
Initiated by interrupt flag set 3-40
Initiated by program exit 3-40
Exchange sequence issue conditions 3-41
Exchange package management 4-42
MEMORY FIELD PROTECTION 3-43
BA register 3-44
LA register 3-44
DEAD START SEQUENCE 3-44

4. INSTRUCTIONS 4-1

INSTRUCTION FORMAT 4-1
Arithmetic, logical format 4-1
Shift, mask format 4-2
Immediate constant format 4-2
Memory transfer format 4-3
Branch format 4-4
SPECIAL REGISTER VALUES 4-5
INSTRUCTION ISSUE 4-5
INSTRUCTION DESCRIPTIONS 4-6
000000 Error exit 4-7
001ijk Monitor functions 4-8
0020xk Transmit (AK) to VL 4-10

0021xx	Set the floating point mode flag in the M register	4-11
0022xx	Clear the floating point mode flag in the M register	4-11
003xjx	Transmit (Sj) to vector mask	4-12
004xxx	Normal exit	4-13
005xjk	Branch to (Bjk)	4-14
006ijkm	Branch to ijkm	4-15
007ijkm	Return jump to ijkm; set B_{00} to (P)	4-16
010ijkm	Branch to ijkm if $(A_0) = 0$	4-17
011ijkm	Branch to ijkm if $(A_0) \neq 0$	4-17
012ijkm	Branch to ijkm if (A_0) positive	4-17
013ijkm	Branch to ijkm if (A_0) negative	4-17
014ijkm	Branch to ijkm if $(S_0) = 0$	4-18
015ijkm	Branch to ijkm if $(S_0) \neq 0$	4-18
016ijkm	Branch to ijkm if (S_0) positive	4-18
017ijkm	Branch to ijkm if (S_0) negative	4-18
020ijkm	Transmit jkm to Ai	4-19
021ijkm	Transmit complement of jkm to Ai	4-19
022ijk	Transmit jk to Ai	4-20
023ijx	Transmit (Sj) to Ai	4-21
024ijk	Transmit (Bjk) to Ai	4-22
025ijk	Transmit (Ai) to Bjk	4-22
026ijx	Population count of (Sj) to Ai	4-23
027ijx	Leading zero count of (Sj) to Ai	4-24
030ijk	Integer sum of (Aj) and (Ak) to Ai	4-25
031ijk	Integer difference (Aj) and (Ak) to Ai	4-25
032ijk	Integer product of (Aj) and (Ak) to Ai	4-26
033ijk	Transmit I/O status to Ai	4-27
034ijk	Block transfer (Ai) words from memory starting at address (A_0) to B register starting at register jk	4-29
035ijk	Block transfer (Ai) words from B registers starting at register jk to memory starting at address (A_0)	4-29
036ijk	Block transfer (Ai) words from memory starting at address (A_0) to T registers starting at register jk	4-29
037ijk	Block transfer (Ai) words from T registers starting at register jk to memory starting at address (A_0)	4-29

040ijkm	Transmit jkm to Si	4-31
041ijkm	Transmit complement of jkm to Si	4-31
042ijk	Form 64-jk bits of one's mask in Si from right	4-32
043ijk	Form jk bits of one's mask in Si from left	4-32
044ijk	Logical product of (Sj) and (Sk) to Si	4-33
045ijk	Logical product of (Sj) and complement of Sk to Si	4-33
046ijk	Logical difference of (Sj) and (Sk) to Si	4-33
047ijk	Logical difference of (Sk) and complement of (Sk) to Si	4-33
050ijk	Scalar merge	4-33
051ijk	Logical sum of (Sj) and (Sk) to Si	4-33
052ijk	Shift (Si) left jk places to S_0	4-36
053ijk	Shift (Si) right 64-jk places to S_0	4-36
054ijk	Shift (Si) left jk places to Si	4-36
055ijk	Shift (Si) right 64-jk places to Si	4-36
056ijk	Shift (Si) and (Sj) left by (Sk) places to Si	4-37
057ijk	Shift (Sj) and (Si) right by (Ak) places to Si	4-37
060ijk	Integer sum of (Sj) and (Sk) to Si	4-38
061ijk	Integer difference of (Sj) and (Sk) to Si	4-38
062ijk	Floating sum of (Sj) and (Sk) to Si.	4-39
063ijk	Floating difference of (Sj) and (Sk) to Si	4-39
064ijk	Floating product of (Sj) and (Sk) to Si	4-40
065ijk	Half-precision rounded floating product of (Sj) and (Sk) to Si	4-40
066ijk	Rounded floating product of (Sj) and (Sk) to Si	4-40
067ijk	Reciprocal iteration; 2-(Sj)*(Sk) to Si	4-40
070ijx	Floating reciprocal approximation of (Sj) to Si	4-42
071ijk	Transmit (Ak) or normalized floating point constant to Si	4-43
072ixx	Transmit (RTC) to Si	4-45
073ixx	Transmit (VM) to Si	4-45
074ijk	Transmit (Tjk) to Si	4-45
075ijk	Transmit (Si) to Tjk	4-45
076ijk	Transmit (Vj element (Ak)) to Si	4-46
077ijk	Transmit (Sj) to Vi element (Ak)	4-46

10hijkm	Read from ((Ah) + jkm) to Ai	4-47
11hijkm	Store (Ai) to (Ah) + jkm	4-47
12hijkm	Read from ((Ah) + jkm) to Si	4-47
13hijkm	Store (Si) to (Ah) + jkm	4-47
140ijk	Logical products of (Sj) and (Vk elements) to Vi elements	4-49
141ijk	Logical products of (Vj elements) and (Vk elements to Vi elements	4-49
142ijk	Logical sums of (Sj) and (Vk elements) to Vi elements	4-49
143ijk	Logical sums of (Vj elements) and (Vk elements) to Vi elements	4-49
144ijk	Logical differences of (Sj) and (Vk elements) to Vi elements	4-49
145ijk	Logical differences of (Vj elements) and (Vk elements) to Vi elements	4-49
146ijk	If VM bit = 1, transmit (Sj) to Vi elements If VM bit ≠ 1, transmit (Vk elements) to Vi elements	4-49
147ijk	If VM bit = 1, transmit (Vj elements) to Vi elements If VM bit ≠ 1, transmit (Vk elements) to Vi elements	4-49
150ijk	Single shift of (Vj elements) left by (Ak) places to Vi elements	4-53
151ijk	Single shift of (Vi elements) right by (Ak) places to Vi elements	4-53
152ijk	Double shifts of (Vj elements) left (Ak) places to Vi elements	4-54
153ijk	Double shifts of (Vj elements) right (Ak) places to Vi elements	4-54
154ijk	Integer sums (Sj) and (Vk elements) to Vi elements	4-59
155ijk	Integer sums (Vj elements) and (Vk elements) to Vi elements	4-59
156ijk	Integer differences of (Sj) and (Vk elements) to Vi elements	4-59
157ijk	Integer differences of (Vj elements) and (Vk elements) to Vi elements.	4-59

160ijk	Floating products of (Sj) and (Vk elements) to Vi elements	4-61
161ijk	Floating products of (Vj elements) and (Vk elements) to Vi elements	4-61
162ijk	Half-precision rounded floating products of (Sj) and (Vk elements) to Vi elements	4-61
163ijk	Half-precision rounded floating products of (Vj elements) and (Vk elements) to Vi elements	4-61
164ijk	Rounded floating product of (Sj) and (Vk elements) to Vi elements	4-61
165ijk	Rounded floating product of (Vj elements) and (Vk elements) to Vi elements	4-61
166ijk	Reciprocal iterations; 2 - (Sj) * (Vk elements) to Vi elements	4-61
167ijk	Reciprocal iterations; 2 - (Vj elements) * (Vk elements) to Vi elements	4-61
170ijk	Floating sums of (Sj) and (Vk elements) to Vi elements	4-64
171ijk	Floating sums of (Vj elements) and (Vk elements) to Vi elements	4-64
172ijk	Floating differences of (Sj) and (Vk elements) to Vi elements	4-64
173ijk	Floating differences of (Vj elements) and (Vk elements) to Vi elements	4-64
174ijx	Floating point reciprocal approximation of (Vj elements) to Vi elements	4-66
175xjk	Test (Vj elements) and enter test results into VM; the type of test made is defined by k	4-68
176ixk	Transmit (VL) words from memory to Vi elements starting at memory address (A_0) and incrementing by (Ak) for successive addresses	4-70
177xjk	Transmit (VL) words from Vj elements to memory starting at memory address (A_0) and incrementing by (Ak) for successive addresses	4-70

5. MEMORY SECTION 5-1

INTRODUCTION 5-1
MEMORY CYCLE TIME 5-1
MEMORY ACCESS 5-1
MEMORY ORGANIZATION 5-3
MEMORY ADDRESSING 5-3
8-BANK PHASING OPTION 5-4
MEMORY PARITY ERROR CORRECTION 5-5

6. INPUT/OUTPUT SECTION 6-1

MEMORY ACCESS 6-1
RESYNCHRONIZATION 6-3
MEMORY BANK CONFLICTS 6-3
I/O MEMORY REQUEST CONDITIONS 6-4
I/O LOCKOUT 6-4
I/O INTERRUPTS 6-4
CHANNEL ERROR CONDITIONS 6-4
I/O MEMORY ADDRESSING 6-5
I/O CHANNEL PARITY 6-5
INPUT CHANNEL SIGNALS 6-5
Input data 6-5
Input ready 6-5
Input resume 6-6
Input disconnect 6-6
OUTPUT CHANNEL SIGNALS 6-6
Output data 6-6
Output ready 6-6
Output resume 6-6
Output disconnect 6-6
CHANNEL OPERATION CONTROL 6-6
I/O CHANNEL MASTER CLEAR 6-7
REAL-TIME CLOCK 6-8

APPENDIXES

A TIMING SUMMARY A-1
B MODULE TYPES B-1
C SOFTWARE CONSIDERATIONS C-1
D INSTRUCTION SUMMARY D-1

FIGURES

1-1 Basic computer system 1-2
2-1 Physical organization of the mainframe 2-2
2-2 General chassis layout 2-3
2-3 Clock pulse waveform 2-7
3-1 Computation section 3-2
3-2 Integer data formats 3-19
3-3 Floating point data format 3-20
3-4 49-bit floating point addition 3-23
3-5 Floating point multiply pyramid 3-25
3-6 Relationship of instruction buffers and registers 3-30
3-7 Instruction buffers 3-33
3-8 Exchange package 3-37
4-1 General format for instructions 4-1
4-2 Format for arithmetic and logical instructions 4-2
4-3 Format for shift and mask instructions 4-2
4-4 Format for immediate constant instructions 4-3
4-5 Format for memory transfer instructions 4-4
4-6 Two-parcel format for branch instructions 4-4
5-1 Memory organization 5-2
5-2 Memory address 5-3
6-1 Channel I/O control 6-2

TABLES

1-1 Characteristics of CRAY-1 Computer System 1-3
2-1 Characteristics of a DD-19 Disk Storage Unit 2-13

SECTION 1

INTRODUCTION

INTRODUCTION 1

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 initialization and for monitoring system performance. The mass storage subsystem provides secondary storage and consists of one to eight 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 so that a maximum mass storage configuration could hold 9.7×10^9 8-bit characters.

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.

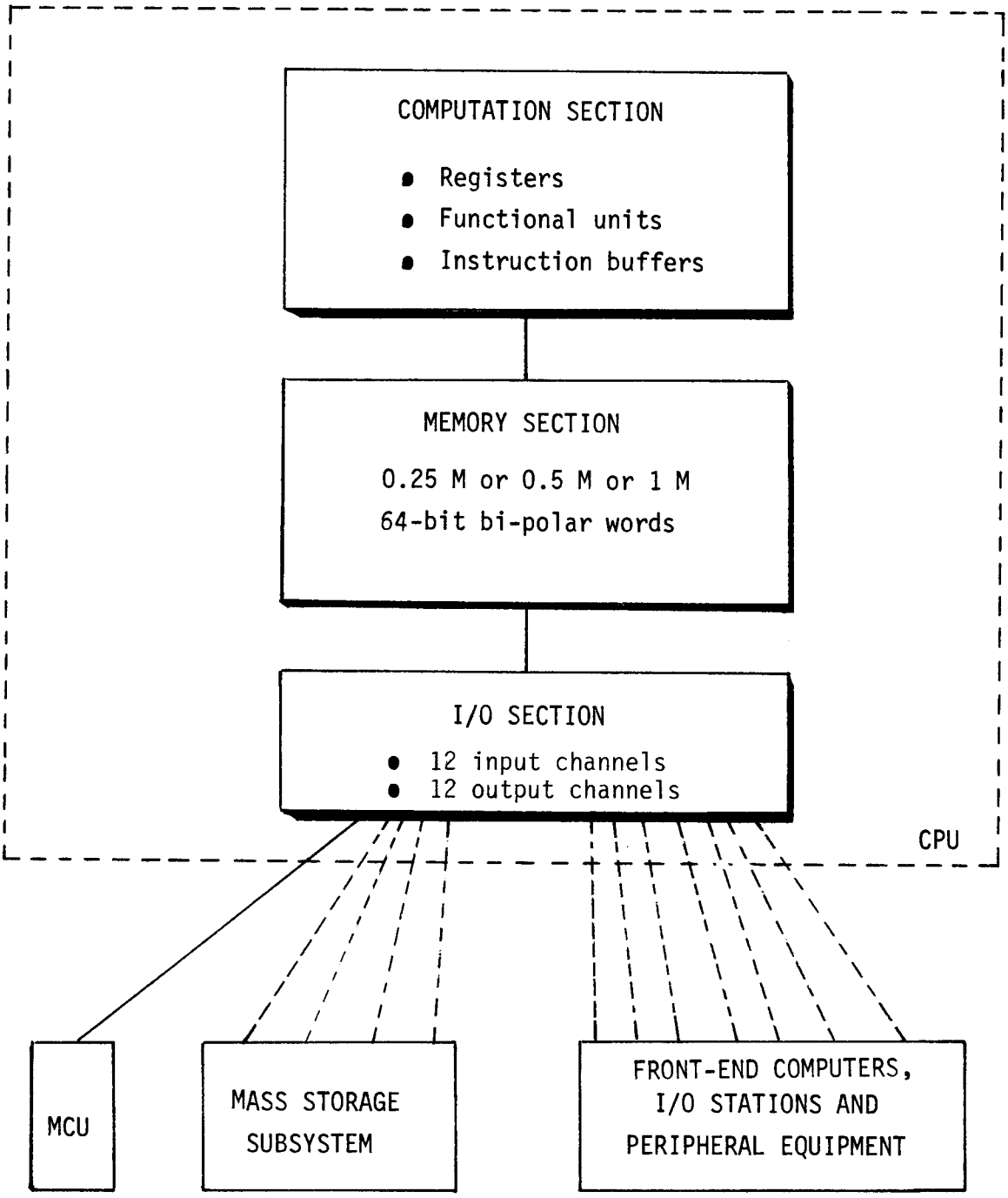


Figure 1-1. Basic computer system

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 of 65,536 words each
- 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 (SEC-DED)

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, and 4 parity bits
- 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, multiplication, 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 1024-bit 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 (132.5 nsec). There is no inherent memory degradation for 16-bank memories of less than one million words.

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

PHYSICAL ORGANIZATION 2

INTRODUCTION

The CRAY-1 computer system consists of the following:

- The CPU mainframe
- A power cabinet
- Two condensing units
- Two motor generators and control cabinets
- A maintenance control unit (MCU)
- One or more disk systems
- An interface to a front-end computer

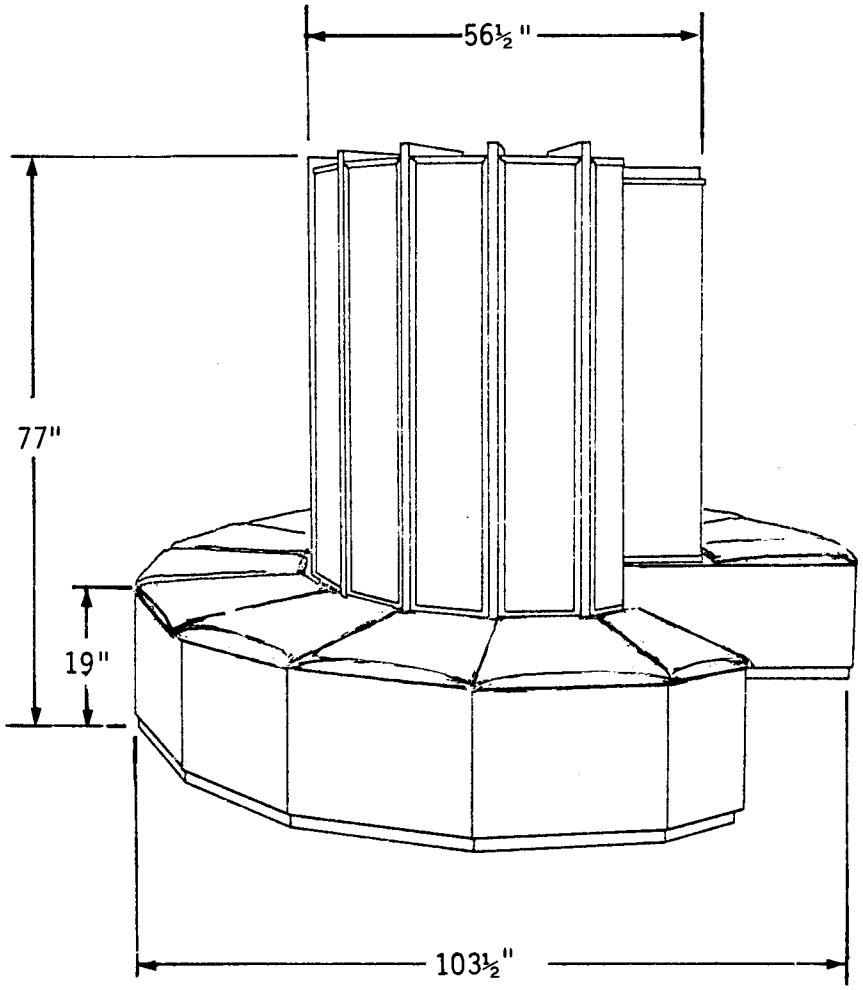
MAINFRAME

The CRAY-1 mainframe, figure 2-1, is composed of 24 logic chassis. The chassis are arranged two per column in a 270^{o} arc which is 56.5 inches in diameter. The twelve columns are 77 inches high. At the base of the columns, 19 inches high and extending outward 30 inches, 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 throughout 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 - 103½ inches diameter by 19 inches high

  Columns - 56½ inches diameter by 77 inches high including height of base

- 24 chassis
- 1662 modules (16 banks); 113 module types
- Each module contains up to 288 IC packages per module
- Power consumption approximately 115 kw input for maximum memory size
- Freon cooled with Freon/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

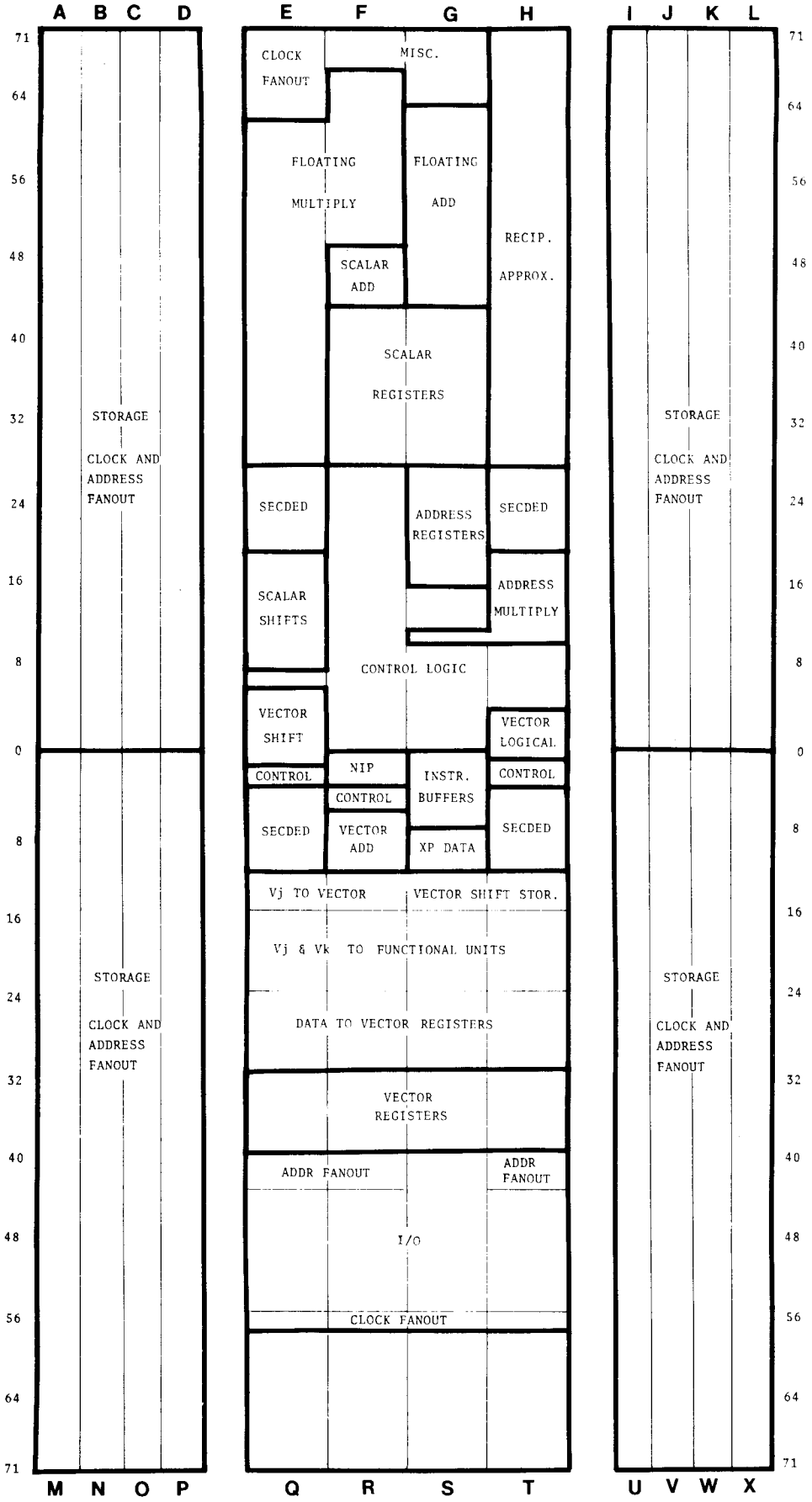


Figure 2-2. General chassis layout

There are 1662 modules in a standard 16-bank† CRAY-1 memory. Modules are arranged 72 per chassis as illustrated in figure 2-2. There are 113 module types. Usage varies from 1 to 708 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 Freon. Power dissipation depends on module density. The maximum module power dissipation by type is approximately 65 watts. The average module dissipation by usage is approximately 49 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 per cent.

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

Printed circuit board

The printed circuit board used in the CRAY-1 computer system is a 5-layer board. The two outer surfaces of the PC board are used for signal runs; the inner three layers are used for the -5.2 V, -2.0 V, and ground supplies. Signal foil runs are a nominal 0.0075 inch. The spacing of the signal layer to the adjacent voltage is a nominal 0.008 inch. The dimensions used provide signal lines with an impedance of 50 to 60 ohms.

Conventional PC techniques are used in the construction of the PC board.

† Refer to 8-Bank Phasing Option, section 5.

Holes are drilled in the PC board for component mounting, interconnecting signal layers, and supplying signal and voltages to components. All holes are plated. The two signal layers are tin-lead plated before etching. The finished PC board is reflowed to eliminate slivers caused by the etching process.

Module assembly

The individual boards of the module are arranged, flow soldered, and inspected prior to being assembled as a module. Logic testing is done at the module level.

Integrated circuit packages

All integrated circuit devices used in the CRAY-1 are packaged in a common package type. The package is a 16-pin hermetically sealed flat pack. Gold or tin-lead plated leads are used depending on the vendor. The 16-pin flat pack was chosen for its reliability and compactness.

IC high-speed logic gate

With minor exceptions, one type of logic gate is used for the central processing unit. This is an ECL circuit with either four or five inputs and with both normal and inverted outputs available to drive loads. One four-input gate and one five-input gate are packaged in a 16-pin flat pack (5/4 gate). All latches, adders, subtracters, etc., are made of this basic gate. The high-speed logic gate has a minimum propagation delay of 0.5 nsec and a maximum propagation delay of 1 nsec. Edge speeds are 1 nsec or less.

IC slow-speed logic gate

The slow-speed gate is a MECL 10K version of the high-speed gate and is used in the memory module for address fanout. The speed is adequate for this application and the lower power requirement is an advantage.

16x1 register chip

The 16x1 register chip provides very fast temporary storage for scalar and vector functional units. The chips are used for instruction buffers and for B, T, and V registers. The chips have a 6 nsec read/write time, well within the 12.5 nsec clock period.

1024x1 memory chip

The bipolar 1024x1 LSI chip is the basic building block around which the CRAY-1 memory is built. The chip was developed by Fairchild using the isoplanar technology. The memory chip has a maximum 50 nsec read/write cycle time. Address decoding is internal to the package and is compatible with standard ECL logic levels.

Resistors

Only two resistor types are used throughout the entire CRAY-1 computer system. They are a center-tapped 120-ohm resistor providing two 60-ohm resistors per package; and a 300-ohm resistor tapped to provide a 120-ohm and 180-ohm resistor. The basic resistor package is a three-lead device in a ceramic substrate. The resistance film is tantalum nitride. The lead frame is thermal pulse bonded. An epoxy covering is used to protect the film from mechanical damage.

All printed circuit boards lines are treated as transmission lines. To provide the proper termination of the transmission lines, each line is parallel-terminated to the -2.0 volt supply. A 60-ohm resistor is used to match the transmission line impedance. To minimize noise on the -2.0 V supply, all used logic gate inputs and outputs are terminated with a 60-ohm resistor to -2.0 volts.

The 16x1 register chip and the 1024x1 memory chip provide only a normal signal output (logic gates provide the normal and inverted output signals). To minimize the noise that could be introduced on the -2.0 volt bus by an unbalanced load, these two devices are terminated with a Thévenin equivalent to the -5.2 volt supply. The 300-ohm resistor is used for the Thévenin equivalent termination.

Connector strips

The module connector strip uses 96 individual sockets molded in plastic. The chassis connector strip uses 96 mating pins molded in plastic. Individual pins and sockets when assembled are mounted on 0.050-inch centers with mounting holes provided in the assembled plastic strip. Each board has 96 holes provided for connecting signals to the module connector

strip. The chassis connector strip is assembled with an 18-inch wire crimped to each pin. Wire pairs are twisted after assembly to provide the twisted pair wire transmission lines. The interconnection of twisted pair wires is made in the center of the line using a solder sleeve.

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.

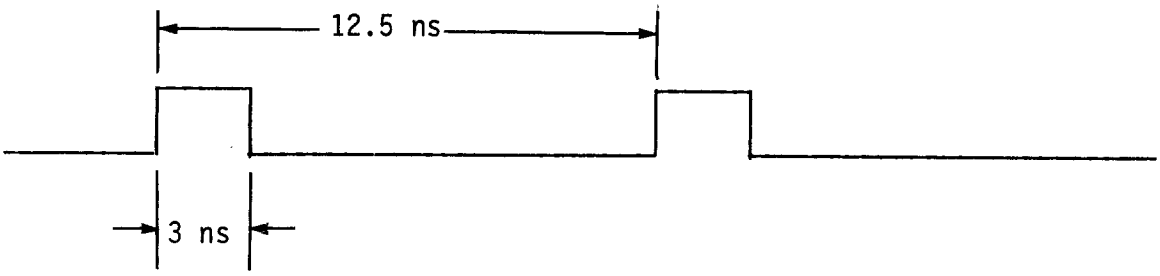


Figure 2-3. Clock pulse waveform

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 Freon-cooled heat sink. Power is distributed via bus bars to the load.

PRIMARY POWER SYSTEM

The primary power system consists of a 150 KW motor generator, motor-generator control cabinet, and power distribution cabinet. The motor generator supplies 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 cold bar which is Freon cooled. 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 128 modules. The cold bar is a cast aluminum bar containing a stainless steel refrigerant tube.

To assure component reliability, the cooling system was designed to provide a maximum case temperature of 130^{o} F (54^{o} C). To meet this goal, the following temperature differentials are encountered:

IC case temperature at center of module	130^{o} F (54^{o} C)
IC case temperature at edge of module	118^{o} F (48^{o} C)
Cold plate temperature at wedge	78^{o} F (25^{o} C)
Cold bar temperature	70^{o} F (21^{o} C)
Refrigerant tube temperature	70^{o} F (21^{o} C)

Two 20-ton compressors are located external to the computer room to complete the cooling system.

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 S-200 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 is interfaced to front-end systems through special interface controllers that compensate for differences in channel widths, machine word size, electrical logic levels, and control protocols. The interface is a Cray Research, Inc. product implemented in ECL logic compatible with the host system. One or more interface controllers are contained in a small chassis located near the CRAY-1 mainframe. 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. It is desirable that channel cables be limited to less than 75 feet. If site conditions require that the interconnected systems be physically located a considerable distance from each other, the effective transmission rate may be degraded.

MASS STORAGE SUBSYSTEM

Mass storage for the CRAY-1 computer system consists of two 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 Freon cooled cabinet located near the mainframe. Up to four controllers may be contained in one cabinet. The cabinet requires about five square feet 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 on a function-by-function basis.

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

COMPUTATION SECTION 3

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 multi-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.

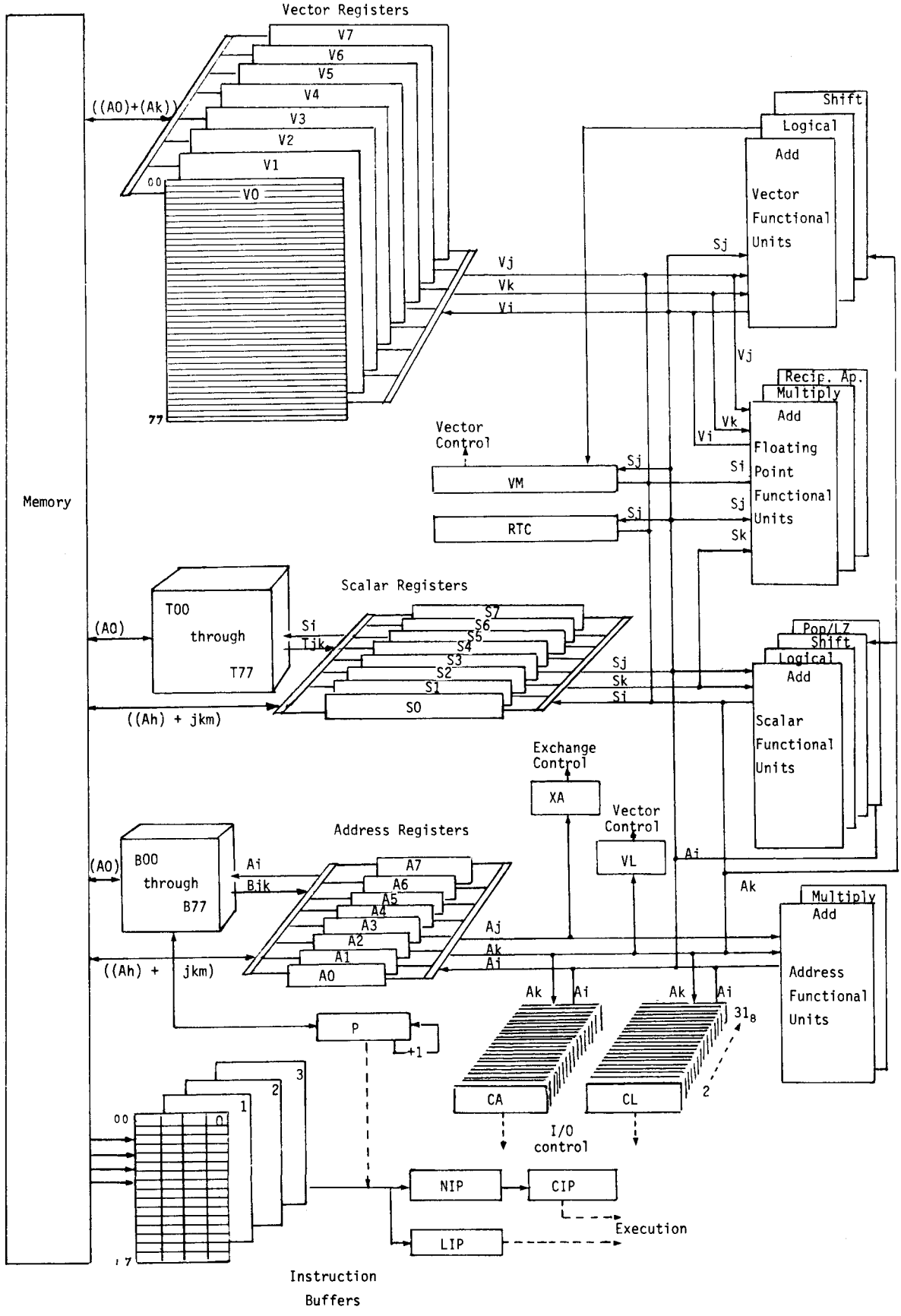


Figure 3-1. Computation section

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 intermediate 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, an increment for the memory address, 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 upon 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 subscript in the range 0 through 7. Vector instructions

reference V registers by allowing specification of the subscript 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.

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. The length of each vector operation is maintained apart from the VL register. Vector operations employing different lengths may proceed concurrently.

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.

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.

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).

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 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 (P) 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 all fully segmented. This means that a new set of operands for unrelated 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 functional unit does not detect overflow of the product.

The functional 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 executes instructions 060 and 061. The addition and subtraction are performed 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 executes 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 executes instructions 042 through 051, the mask and Boolean instructions.

The scalar logical unit is an integral part of the modules containing the S registers. Since data does not have to leave the modules for the function to be performed, operations require only one clock period.

Population/leading zero count unit

This functional unit executes instructions 026 and 027. Instruction 026, which counts the number of bits having a value of one in the operand, executes in four clock periods. Instruction 027, 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.

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:

$$(V2_{00}) = (V2_{00}) + (V1_{00})$$
$$(V2_{01}) = (V2_{00}) + (V1_{01})$$
$$(V2_{02}) = (V2_{00}) + (V1_{02})$$
$$(V2_{03}) = (V2_{00}) + (V1_{03})$$
$$(V2_{04}) = (V2_{00}) + (V1_{04})$$
$$(V2_{05}) = (V2_{00}) + (V1_{05})$$
$$(V2_{06}) = (V2_{00}) + (V1_{06})$$
$$(V2_{07}) = (V2_{00}) + (V1_{07}) \qquad \underbrace{\text{new } (V2_{00})}$$
$$(V2_{08}) = (V2_{00}) + (V1_{08}) = (V2_{00}) + (V1_{00}) + (V1_{08})$$
$$(V2_{09}) = (V2_{01}) + (V1_{09}) = (V2_{00}) + (V1_{01}) + (V1_{09})$$
$$(V2_{10}) = (V2_{02}) + (V1_{10}) = (V2_{00}) + (V1_{02}) + (V1_{10})$$

(contents of register V2, continued)

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

$\vdots$

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

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 executes instructions 154 through 157. The addition and subtraction 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 executes 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.

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 executes 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 060 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 normalized 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. Functional unit time is 14 clock periods.

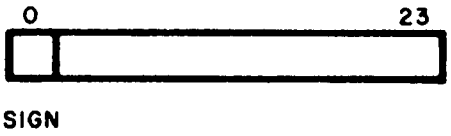
The result is normalized. The input operand is assumed to be normalized; the uppermost bit of the coefficient is not tested but is assumed to be set in the computation.

ARITHMETIC OPERATIONS

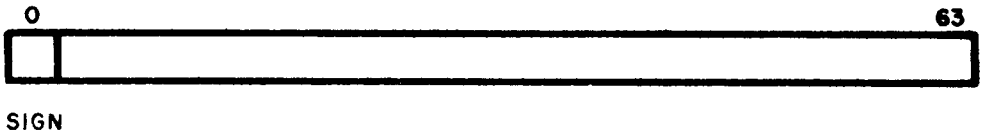
Functional units in the CRAY-1 either perform two's complement integer arithmetic or perform floating point arithmetic.

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.



2's COMPLEMENT INTEGER (24 BITS)



2's COMPLEMENT INTEGER (64 BITS)

Figure 3-2. Integer data formats

Multiplication of two fractional 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.

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^{-2500} through 10^{+2500}.

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 is not in effect. The floating point mode flag can be set or cleared by an object program. The object program has the responsibility to clear the f.p. mode flag via a 0022 instruction at the beginning of each vector branch sequence and resetting it via a 0021 instruction after the merge.

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

Floating point add unit - 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.
```

Floating point multiply unit - In the floating point multiply unit, if the exponent of either operand is greater than or equal to 60000_8 or if the the sum of the two exponents is greater than or equal to 60000 , the floating point error flat 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 result exponent is less than or equal to 17777_8 and causes an all zero exponent and coefficient to be returned to the result register.

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.

Floating point reciprocal approximation unit - For the floating point 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.

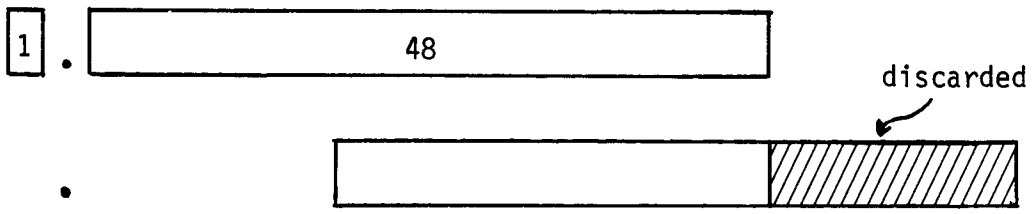


Figure 3-4. 49-bit floating point addition

Multiplication algorithm

The floating point multiply unit in the CRAY-1 computer 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.

Considering that the binary point is to the left of the top bit of the pyramid, that is, at 2^{-1}, the maximum value of the truncated amount is approximately $2^{-59} \times 317 + 2^{-96}$ or about 5.5×10^{-16}. Therefore, the effect of the truncation constant on the entire multiply pyramid is a product variation in the following range:

$$1.16 \times 10^{-16} \text{ to } 6.66 \times 10^{-16}$$

The net result of this on the 48-bit fraction transmitted from the functional unit is the possibility of rounding up 2^{-48}.

In a full precision rounded multiply, a round bit is entered into the pyramid at 2^{-49} and allowed to propagate up the pyramid. However, in this case, the product bit 2^{-49} is forced to zero.

The following table illustrates the effects of multiply truncation and rounding.

Bit	-48	-49	-50	-51	-52	-53	-54	-55	-56	-57	-58	-59	-60	-61	-62	-63	
						3	11	16	21	26	37	36	35	34			Max. missing log. prod.
	1	2	4	9	16	21	26	31	36	35	34	33	32				Missing carry
	1	2	4	9	19	32	42	52	62	72	70	68	66				Bits truncated from bit position
	1	0	0	1	1	0	0	0	0	0	0	0	0				
								5	3								
	1	0	0	1	1	1	1	0	1	0	0	0	0				$= 2^{-59} \times 317 \cong 5.5 \times 10^{-16}$
	256			32	16	8	4		1								
	k	k															$= 2^{-52} \times 3 \cong 6.66 \times 10^{-16}$
R																	$= 2^{-49} \cong 1.776 \times 10^{-15}$

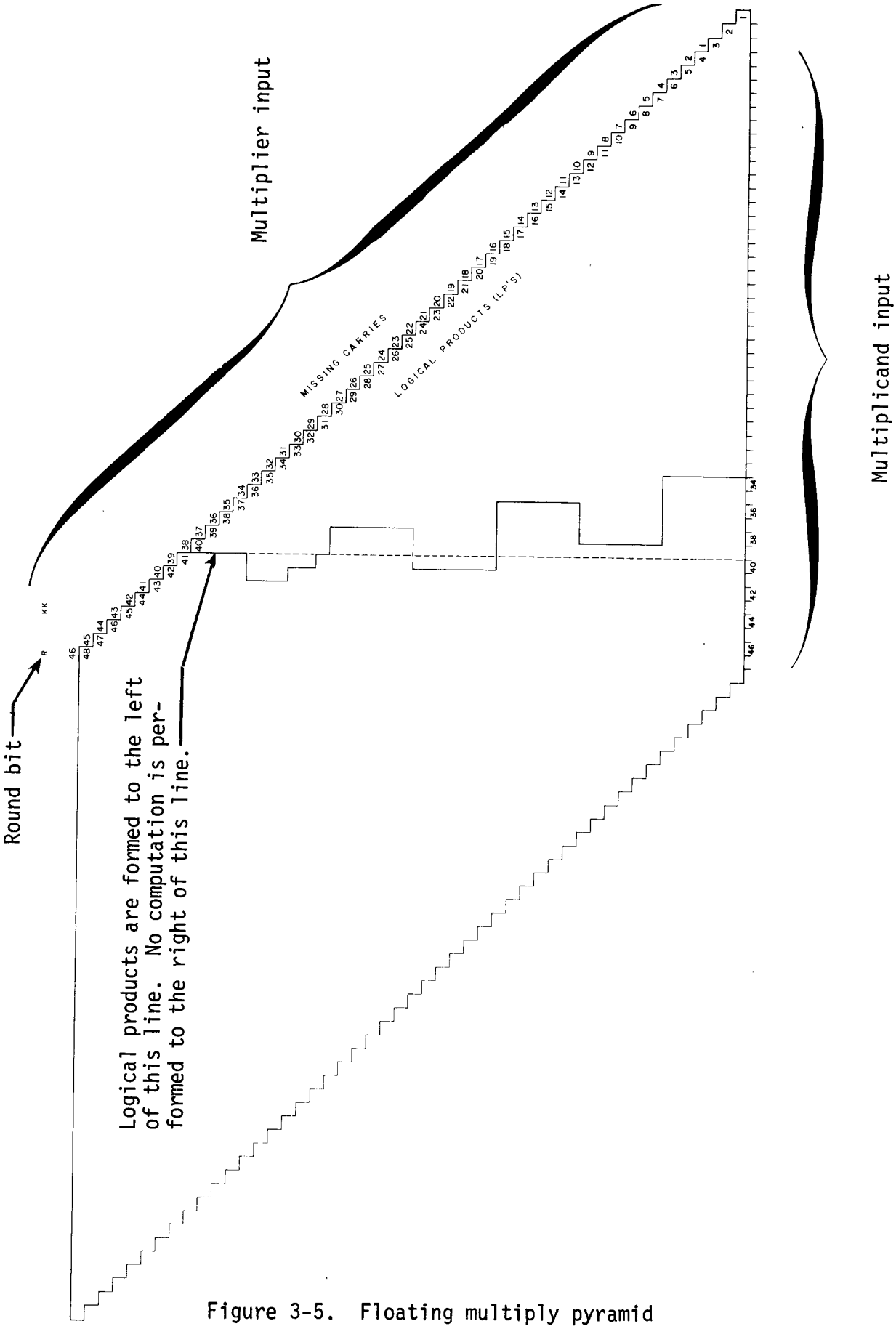


Figure 3-5. Floating multiply pyramid

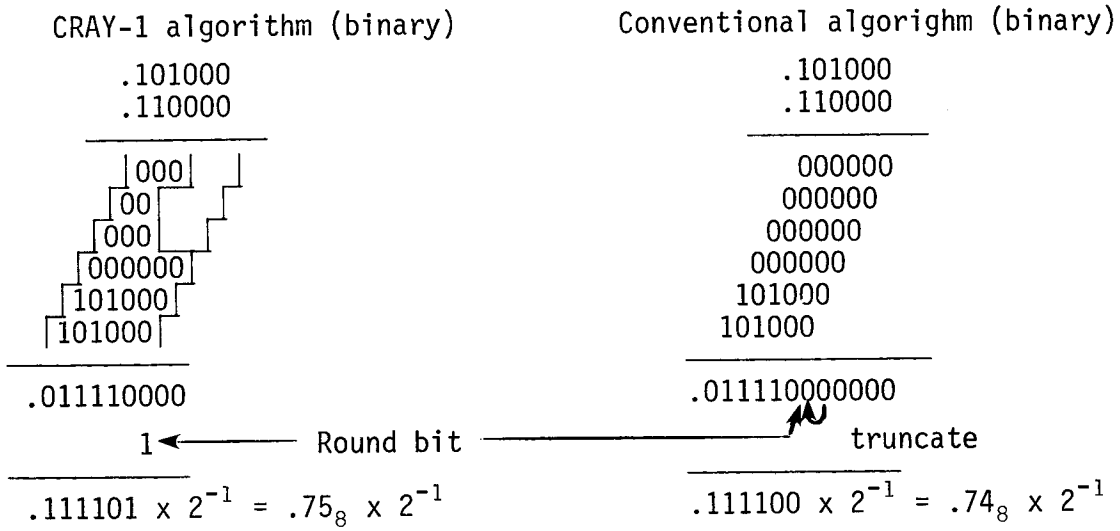
Note that reversing the multiplier and multiplicand operands could cause slightly different results, that is, A x B is not necessarily the same as B x A.

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 31-bit result (2^{-1} to 2^{-30}) is transmitted back. If the result requires a left shift, the bottom bit (2^{-30}) will be zero.

A few simplified examples may help to illustrate the CRAY-1 multiplication algorithm. Each of these examples uses only 6-bit arithmetic to aid understanding of this algorithm and its differences from the conventional algorithm. The multiplication is shown in the usual school presentation (intermediate additions are not shown).

CASE 1 - RESULT FROM CRAY-1 ALGORITHM 1 BIT HIGH

0.5 x 0.6



CASE 2 - CRAY-1 ALGORITHM ROUNDS CORRECTLY AND IS COMMUTATIVE

0.74 x 0.53

CRAY-1 algorithm, both orders of multiplication

```
      .111100          .101010
      .101010          .111100
-----------------  -----------------
        000              000
       11               00
      000              101
     111100           101010
    000000           101010
   111100           101010
-----------------  -----------------
  .100111000         .100111010
        1                  1
-----------------  -----------------
  .100111 = .47₈     .100111 = .47₈
```

Conventional Algorithm

```
      .111100
      .101010
-----------------
       000000
      111100
     000000
    111100
   000000
  111100
-----------------
 .100111011000
-----------------
 .100111 = .47₈
```

CASE 3 - CRAY-1 ALGORITHM NOT COMMUTATIVE, 1 BIT LOW IN ONE ORDER, ROUNDS CORRECTLY IN THE OTHER ORDER.

CRAY-1 algorithm, both orders of multiplication

```
      .101011          .110010
      .110010          .101011
-----------------  -----------------
        000              110
       10               11
      000              000
     000000           110010
    101011           000000
   101011           110010
-----------------  -----------------
  .100001010         .100001100
  1← Round bit →1
    Truncate
-----------------  -----------------
  .100001 = 41₈      .100010 = 42₈
```

Conventional Algorithm

```
      .101011
      .110010
-----------------
       000000
      101011
     000000
    000000
   101011
  101011
-----------------
 .100001100110
        Round bit
-----------------
 .10010 = .42₈
```

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	1 1 0 0
result	1 0 0 0

A logical difference is the exclusive OR function:

operand one	1 0 1 0
operand two	1 1 0 0
result	0 1 1 0

A logical sum is the inclusive OR function:

operand one	1 0 1 0
operand two	1 1 0 0
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.

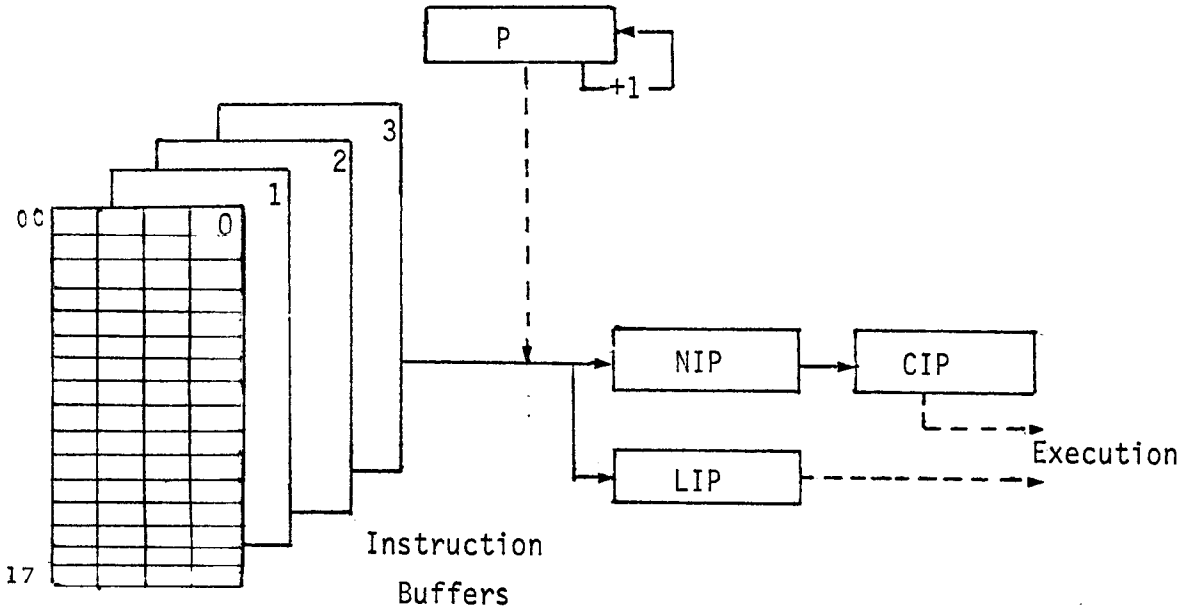


Figure 3-6. Relationship of instruction buffers and registers

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. A noisy 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 a noisy 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.

If issue of the instruction in the CIP register is delayed, the data in the NIP register is held over for the next clock period. Data entry to the NIP register is blocked for the second parcel of a two-parcel instruction. The resulting blank is then issued at the CIP register at the proper time as a do nothing instruction. A blank instruction differs from a 000 instruction only in that a true 000 instruction causes an error interrupt.

Data entry in the NIP register is also blocked when the arriving data is not valid. This occurs on crossing buffer boundaries and on branching as well as on interrupt conditions.

The NIP register is not master cleared. A noisy 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. This register is almost the same as the NIP register except that it contains valid data at times when the same data has been blocked from entering the NIP 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.

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,

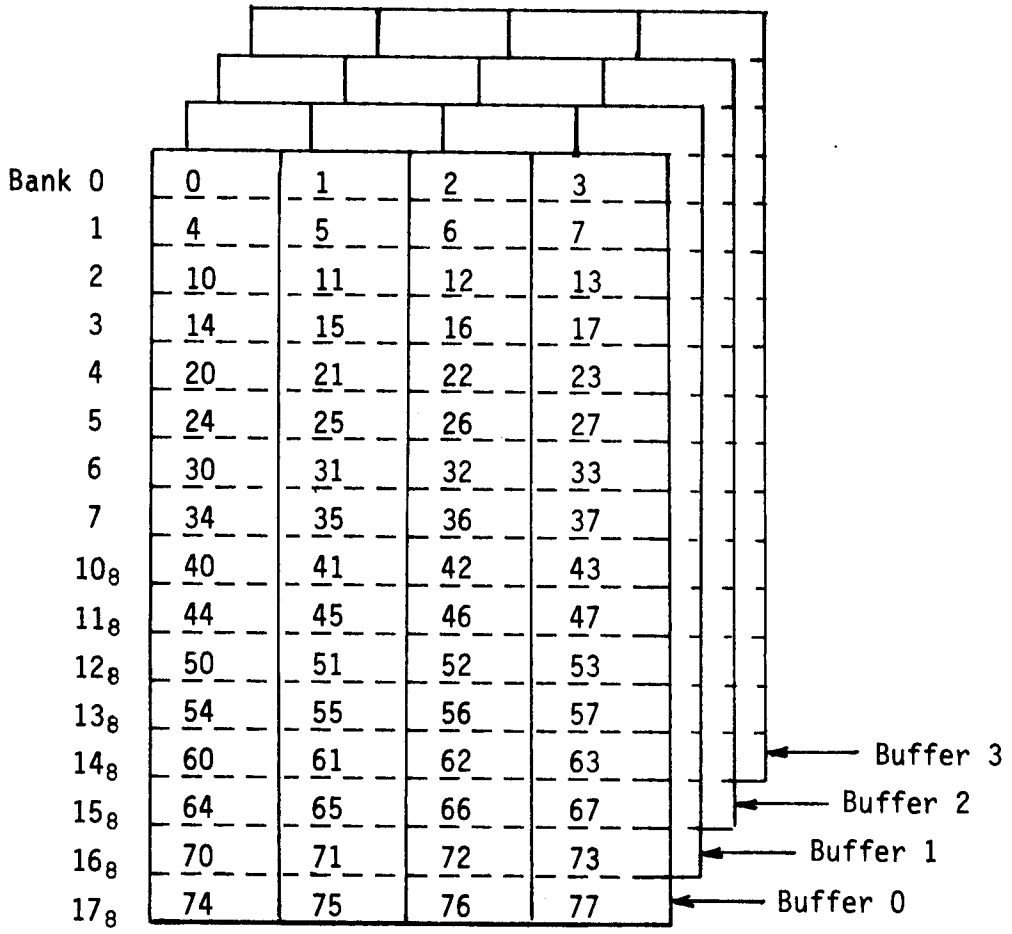


Figure 3-7 Instruction buffers

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.

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 instruction 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 four-bit register that contains part of the exchange package for a currently active program. The four bits are selectively set during an exchange sequence. Bit 37, the floating point error mode flag, can be set or cleared during the execution interval for a program through use of the 0021 and 0022 instructions. The remaining bits are not altered during the execution interval for the exchange package and can be altered only when the exchange package is inactive in storage. Bits are assigned as follows in word two of the exchange package.

- Bit 36 Correctable memory error mode flag. When this bit is set, interrupts on correctable errors are enabled.
- Bit 37 Floating point error mode flag. When this bit is set, interrupts on floating point errors are enabled.
- Bit 38 Uncorrectable memory error mode flag. When this bit is set, interrupts on uncorrectable memory errors are enabled.
- Bit 39 Monitor mode flag. When this bit is set, all interrupts other than memory errors are inhibited.

F REGISTER

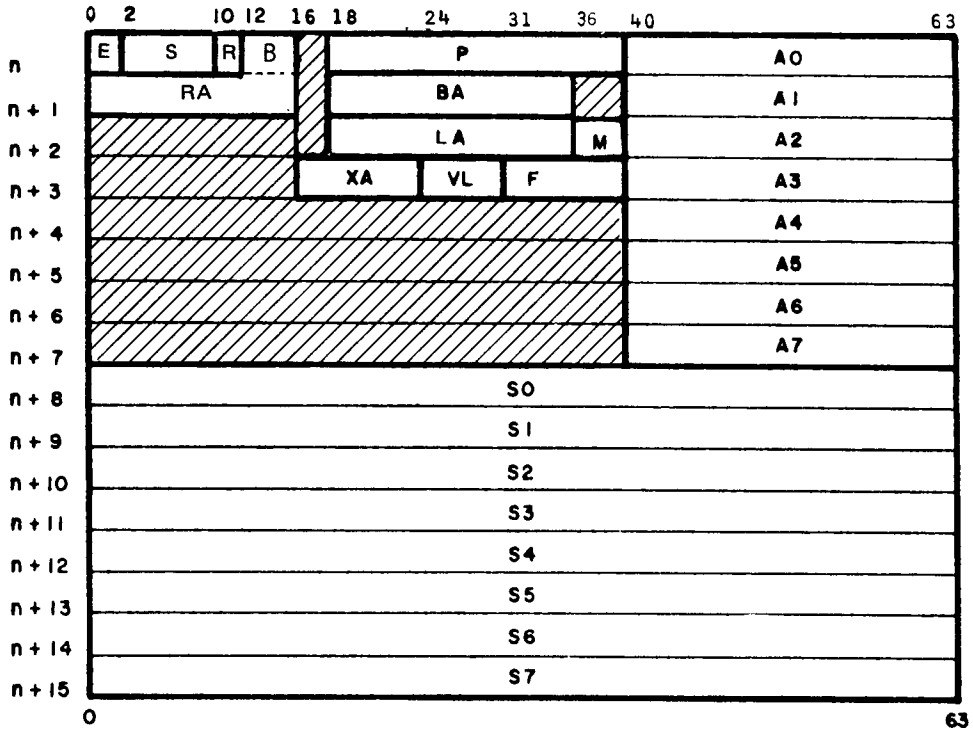
The F (flag) register is a nine-bit register that contains part of the exchange package for the currently active program. This register contains nine flags which are individually identified with the exchange package in figure 3-8. Setting any of these flags causes interruption of the program execution. 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 bits is set, another exchange will occur immediately.

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 are otherwise present, the flag remains cleared and no exchange sequence is 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
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



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)

10 Uncorrectable memory

01 Correctable memory

R - Read mode (bits 10,11)

00 Scalar

01 I/O

10 Vector

11 Fetch

M - Modes†

36 Interrupt on correctable memory error

37 Interrupt on floating point

38 Interrupt on uncorrectable memory error

39 Monitor mode

F - Flags†

31 Console interrupt

32 RTC interrupt

33 Floating point error

34 Operand range

35 Program range

36 Memory error

37 I/O interrupt

38 Error exit

39 Normal exit

†Bit position from left of word

Figure 3-8. Exchange package

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.

Syndrome (S) - 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.

Read mode (R) - 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

Read address (RAB) - 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.

Active exchange package

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 is in effect 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 is the same in either case and the difference is 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 sensed 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

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.

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 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 noisy 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 signal clears the input channel address registers to zero and sets an active status. 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 begin execution of the monitor program. Subsequent actions are dictated by the design of the operating system.

SECTION 4

INSTRUCTIONS

INSTRUCTIONS 4

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:

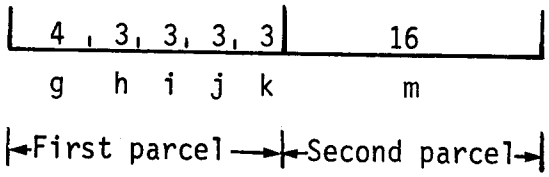


Figure 4-1. General format for instructions

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.

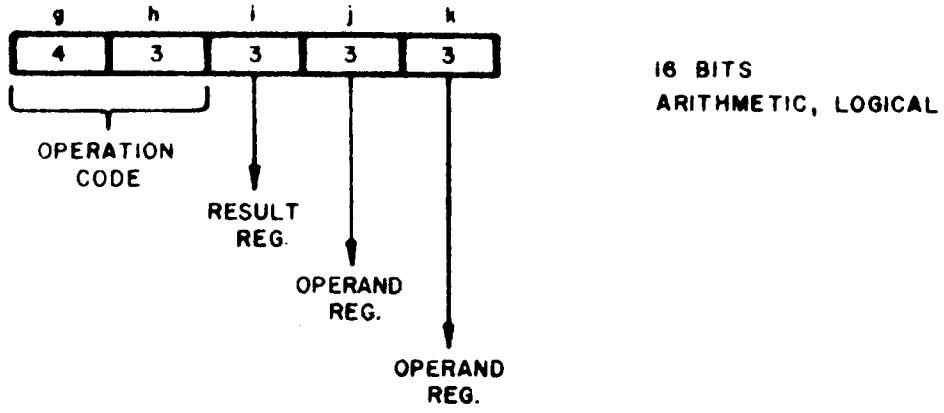


Figure 4-2. Format for arithmetic and logical instructions

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 designates 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.

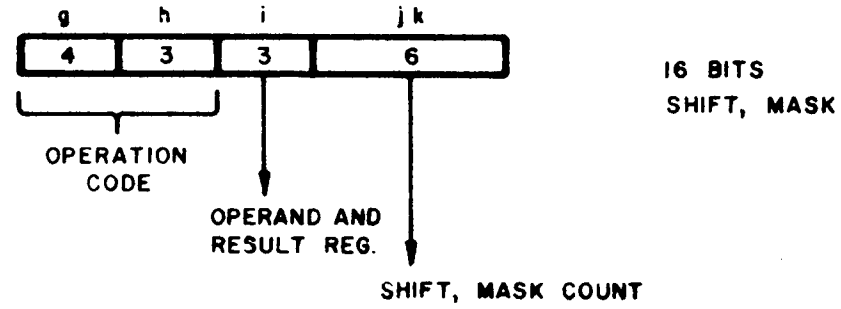


Figure 4-3. Format for shift and mask instructions

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.

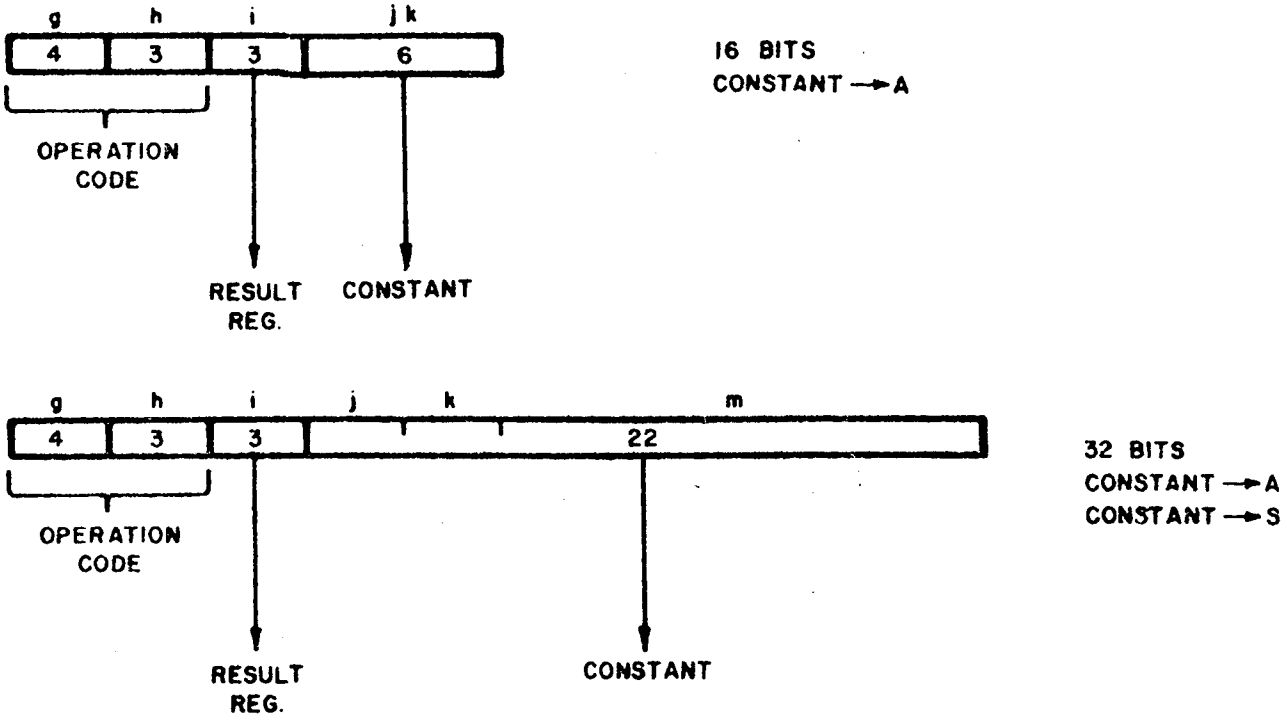


Figure 4-4. Format for immediate constant instructions

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.

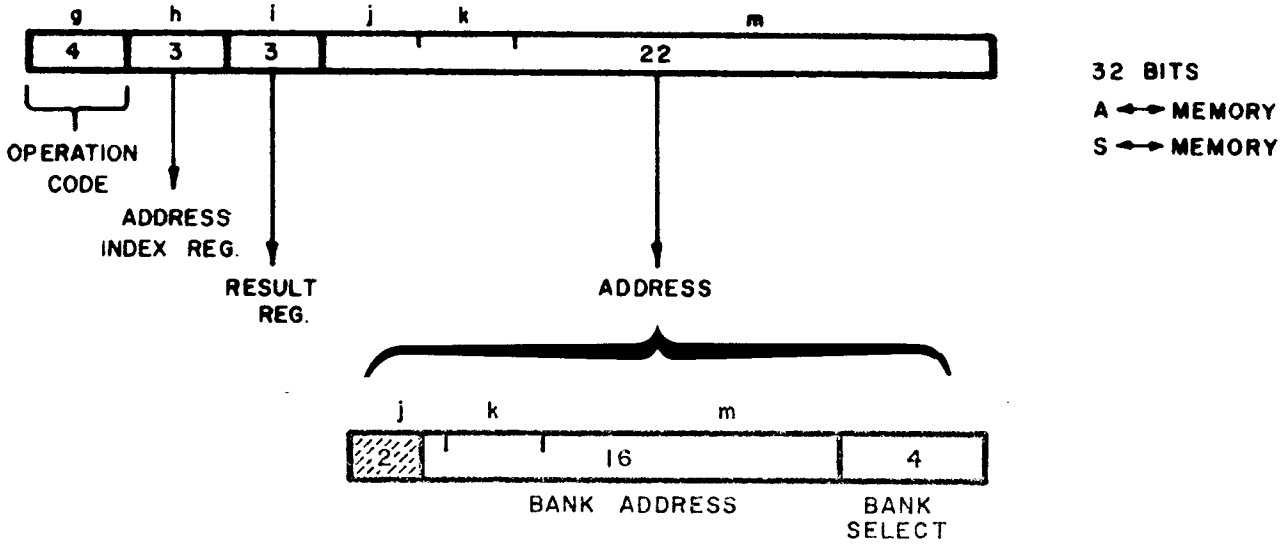


Figure 4-5. Format for memory transfer instructions

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.

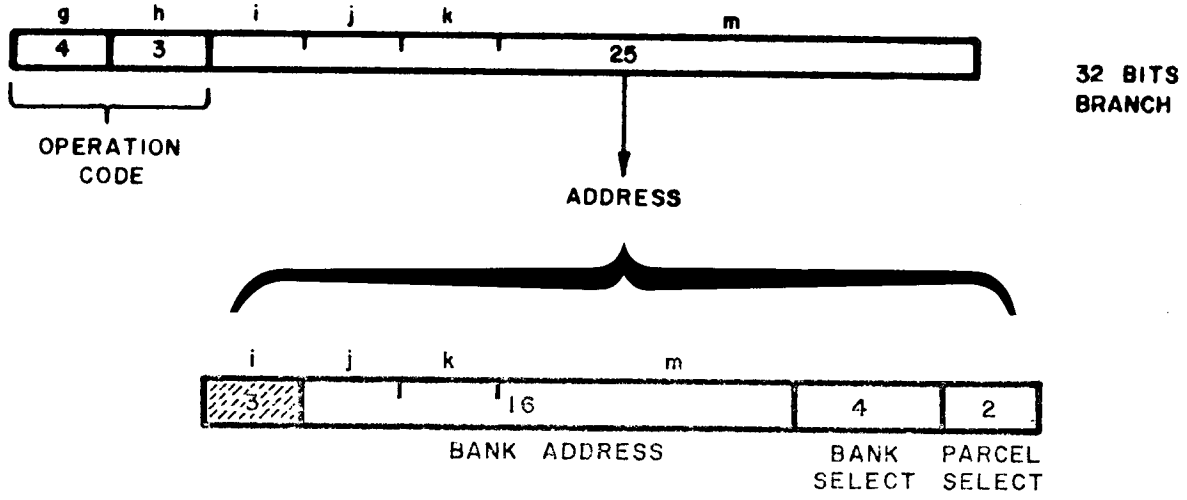


Figure 4-6. Two-parcel format for branch instructions

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.

000000	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

Special cases

None

001ijk	Monitor functions

This instruction 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)
0012jk	Clear the interrupt flag and error flag for the channel indicated by (Aj)
0013jk	Enter the XA register with (Aj)
0014jk	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.

Hold issue conditions

034 - 037 in process
Exchange in process
For 0010 and 0011, Aj or Sj or Ak reserved
For 0012 or 0013, Aj or Sj or Ak reserved
For 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.

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 ≠ 0 and (Ak) = 0

0021xx	Set f.p. mode flag in M register
0022xx	Clear f.p. 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.

The object program has the responsibility of clearing the f.p. mode flag at the beginning of a vector branch sequence and of resetting the flag after the merge.

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

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 $(A_0) = 0$
011ijkm	Branch to ijkm if $(A_0) \neq 0$
012ijkm	Branch to ijkm if (A_0) positive
013ijkm	Branch to ijkm if (A_0) 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

014ijkm	Branch to ijkm if $(S_0) = 0$
015ijkm	Branch to ijkm if $(S_0) \neq 0$
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

026ijx	Population count of (Sj) to Ai

This 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 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 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 ≠ 0
(Ai) = 1 if j = 0 and k = 0
(Ai) = (Aj)+1 if j ≠ 0 and k = 0

For 031:

(Ai) = -(Ak) if j = 0 and k ≠ 0
(Ai) = -1 if j = 0 and k = 0
(Ai) = (Aj)-1 if j ≠ 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.

Hold issue conditions

034 - 037 in process
Exchange in process
A register access conflict
Ai reserved
Aj reserved

Execution time

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/O 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 (A_0) to B registers starting at register jk.
035ijk	Block transfer (Ai) words from B registers starting at register jk to memory starting at address (A_0)
036ijk	Block transfer (Ai) words from memory starting at address (A_0) to T registers starting at register jk
037ijk	Block transfer (Ai) words from T registers starting at register jk to memory starting at address (A_0)

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 (A_0). The A_0 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
       -------
(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
       -------
(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) = 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

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 $\neq$ 0
(Si) = (Sj) normalized if (Sj) exponent is valid, j $\neq$ 0 and k = 0

For 063:

(Si) = -(Sk) normalized if j = 0 and k $\neq$ 0
(Si) = (Sj) normalized if (Sj) exponent is valid, j $\neq$ 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 0 to 9 CPs + 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 27 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 transmitting 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.

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

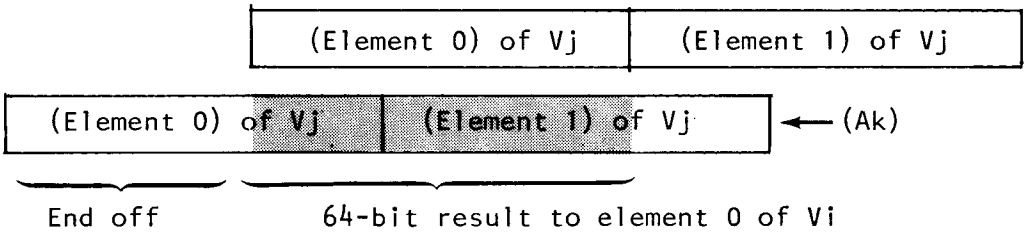
152ijk	Double shifts of (Vj elements) left (Ak) places to Vi elements
153ijk	Double shifts of (Vj elements) right (Ak) places to Vi elements

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.

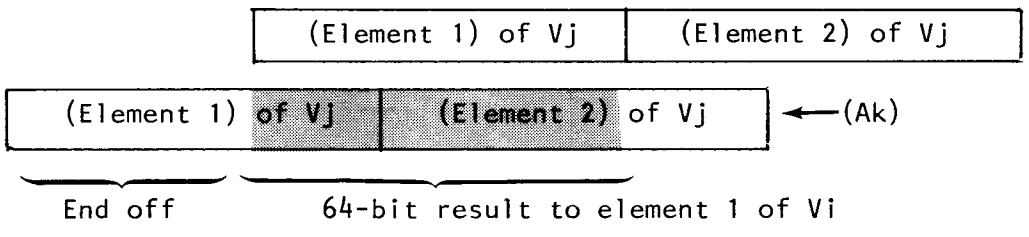
All shifts are end-off with zero fill.

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

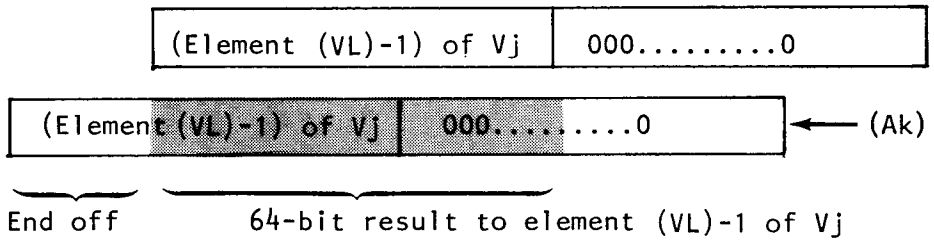
The 152 instruction performs left shifts. In the general case, element 0 of Vj is joined with element 1 and the 128-bit quantity is shifted left by the amount specified by (Ak). The 64 high order bits of the result are transmitted to element 0 of Vi. The figure below illustrates this operation.



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:



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.



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

Example:

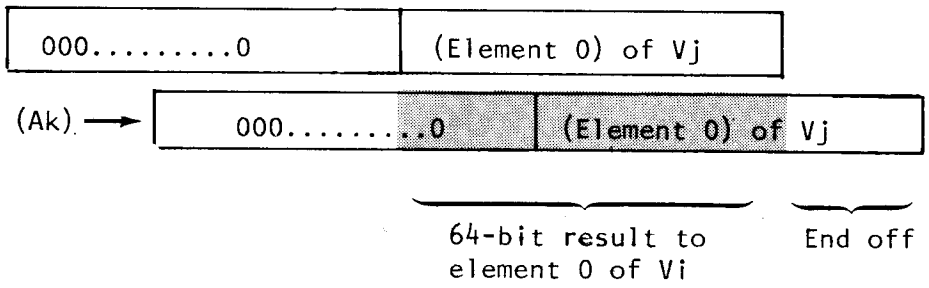
Suppose that a 152 instruction is to be executed and the following register conditions exist:

(VL) = 4
(A1) = 3
(Element 0) of V_4 = 0 00000 0000 0000 0000 0007
(Element 1) of V_4 = 0 60000 0000 0000 0000 0005
(Element 2) of V_4 = 1 00000 0000 0000 0000 0006
(Element 3) of V_4 = 1 60000 0000 0000 0000 0007

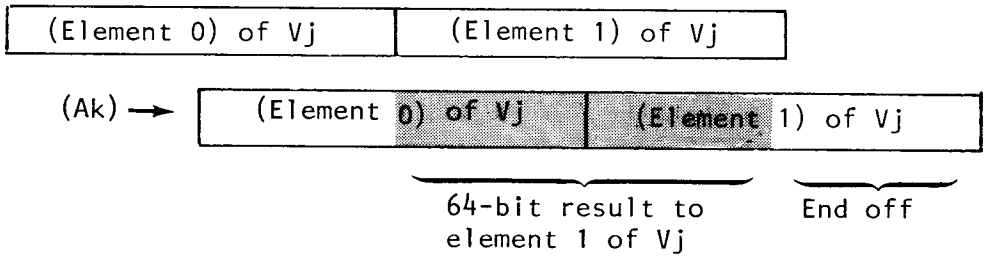
Instruction 152541 is executed and following execution, the first four elements of V_5 contain the following values:

(Element 0) of V_5 = 0 00000 0000 0000 0000 0073
(Element 1) of V_5 = 0 00000 0000 0000 0000 0054
(Element 2) of V_5 = 0 00000 0000 0000 0000 0067
(Element 3) of V_5 = 0 00000 0000 0000 0000 0070

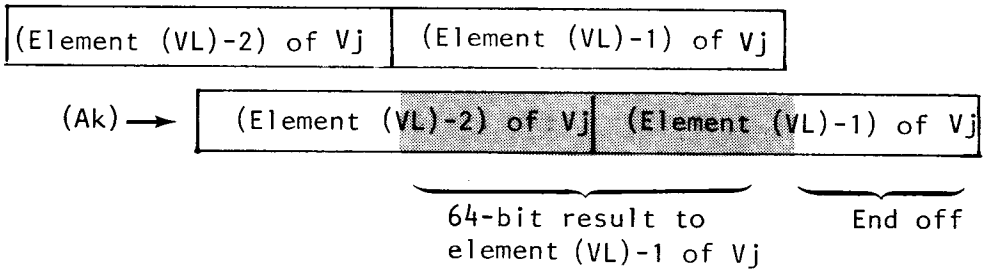
The 153 instruction performs right shifts. Element 0 of Vj is joined with 64 low-order bits of zero and the 128 bit quantity is shifted right by the amount specified by (Ak). The 64 low-order bits of the result are transmitted to element 0 of Vi. The figure below illustrates this operation.



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.



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.



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 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

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 CP
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

Execution time

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

Special cases

(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.

174ijx	Floating point reciprocal approximation 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 27 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 21 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.

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 (A_0) and incrementing by (Ak) for successive addresses
177xjk	Transmit (VL) words from Vj elements to memory starting at memory address (A_0) 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) ≤ 5

Vi ready (VL) + 8 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) ≤ 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

MEMORY SECTION 5

INTRODUCTION

The memory for the CRAY-1 normally consists of 16 banks† of bi-polar 1024-bit LSI memory. Three memory size options 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 (137.5 nsec). There is no inherent memory degradation for 16-bank memories of less than one million words.

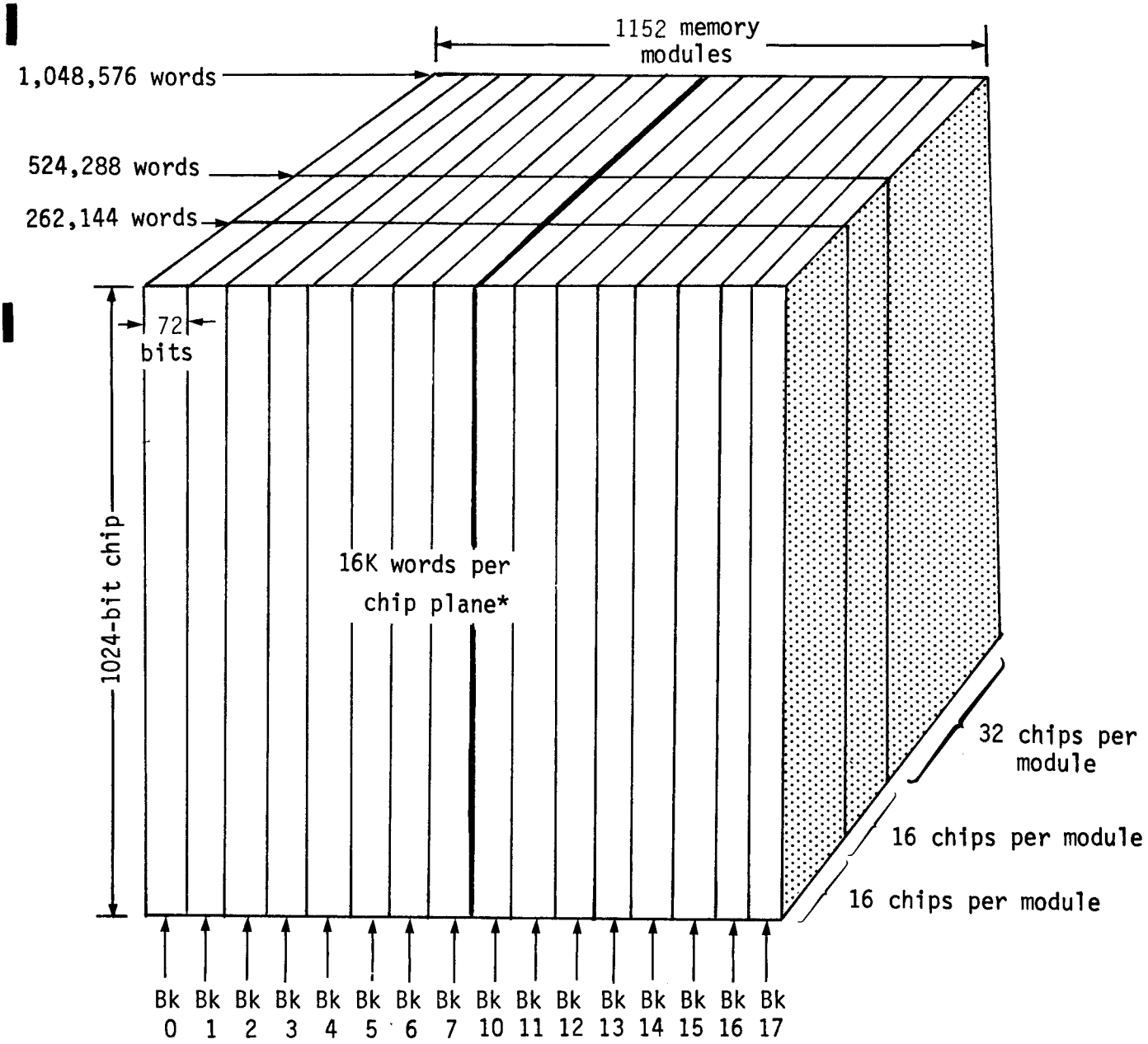
MEMORY ACCESS

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.

Scalar instructions referencing memory have priority over I/O requests arriving at memory in the same clock period. The I/O request is rejected and resubmitted eight clock periods later. Vector references to memory block access for the length of the transfer plus four clock periods. However, the I/O channel requests are rejected during these transfers.

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

† Refer to 8-Bank Phasing Option later in this section.



* A chip plane is one chip per module, 72 modules per bank (64 data; 8 SECDED).

Adding a chip to every module increases memory by 16K words.

Figure 5-1. Memory organization (16 banks)

to clear the I/O channels. Such degradation is anticipated to be less than 10 percent over the same program running with no I/O interference.

MEMORY ORGANIZATION

The memory is organized into 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. For a one-million word memory, each module contains 64 memory chips. For smaller memory configurations, the number of bits per module is reduced by reducing the number of memory chips per module as illustrated in figure 5-1. Each module in a half-million word memory contains 32 memory chips; each module in a quarter-million word memory contains 16 memory chips. This technique allows a reduction in memory size with no sacrifice in performance. I/O characteristics and memory access time are the same for all memory sizes. Memory configurations are field upgradable.

MEMORY ADDRESSING

A word in a 16-bank memory is addressed in 20 bits as shown in figure 5-2.

The low order four bits specify one of the 16 banks.
The next 10 bits specify an address within the chip.
The upper six bits specify one of up to 64 chips on the module.

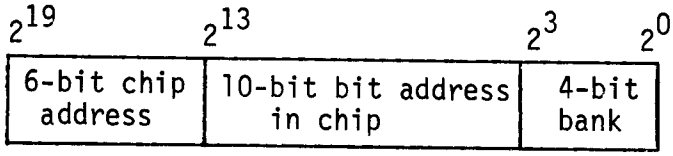


Figure 5-2. Memory address

The address structure provided for by the instruction format is sufficient to address four million words directly; however, there are no plans to incorporate this size memory into the CRAY-1.

† Refer to description of 8-Bank Phasing Option

8-BANK PHASING OPTION

Although 16-bank phasing is standard on the CRAY-1, 8-bank phasing (allowing a maximum memory size of 1/2 million words) can be accomplished by replacing two modules and setting the bank select switch to the left or right banks. This option is available on any 16-bank memory machines for the purpose of maintaining memory since it allows the system to run with half of its memory while the other half is being worked on.

Eight-bank phasing also makes possible a system consisting of one-half million words arranged in only eight banks rather than the conventional sixteen. Such a system could be easily field upgraded to a 16-bank full million words simply by completing the remaining banks.

For 8-bank phasing, each of the 576 modules in a half-million word memory contains 64 memory chips; each module in a quarter-million word memory contains 32 memory chips.

A word in an 8-bank memory is addressed in 19 bits.

The low order three bits specify one of the 8 banks.
The next 10 bits specify an address within the chip.
The upper six bits specify one of the 64 chips on the module.

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 176 and 177 instructions, (Ak) determines speed control. For 8 banks, incrementing by 8 places 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.

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

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 5-3.)

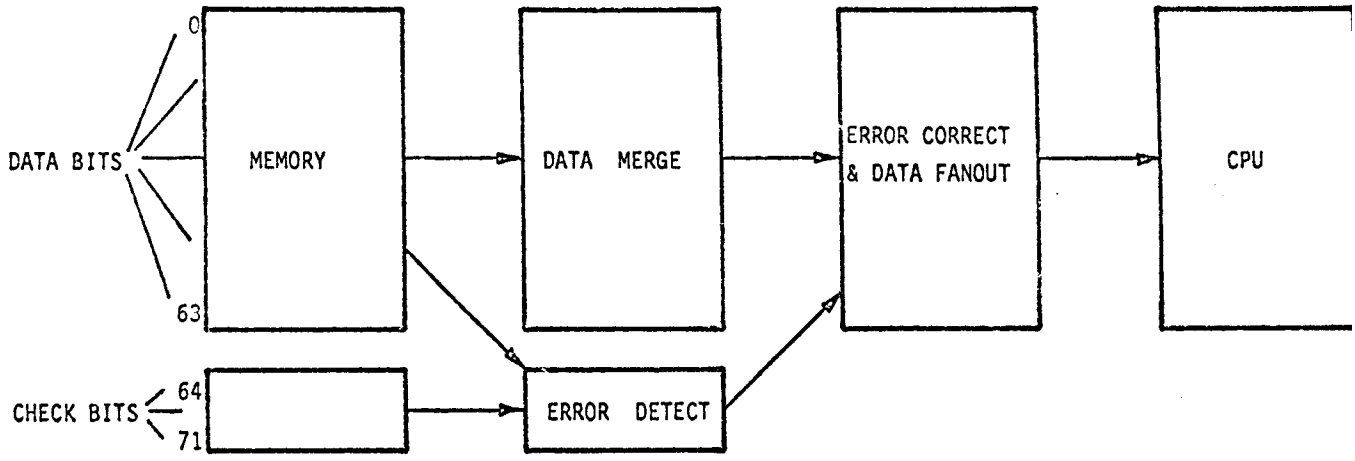


Figure 5-3. Memory data path with SEC-DED.

The network operates on the basis of single error correction, double error detection (SEC-DED). 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.

The SEC-DED 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 5-4 shows the bits of the data word used to determine the state of each check bit. An X in the

---

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

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 odd 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 5-4 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 5-4.

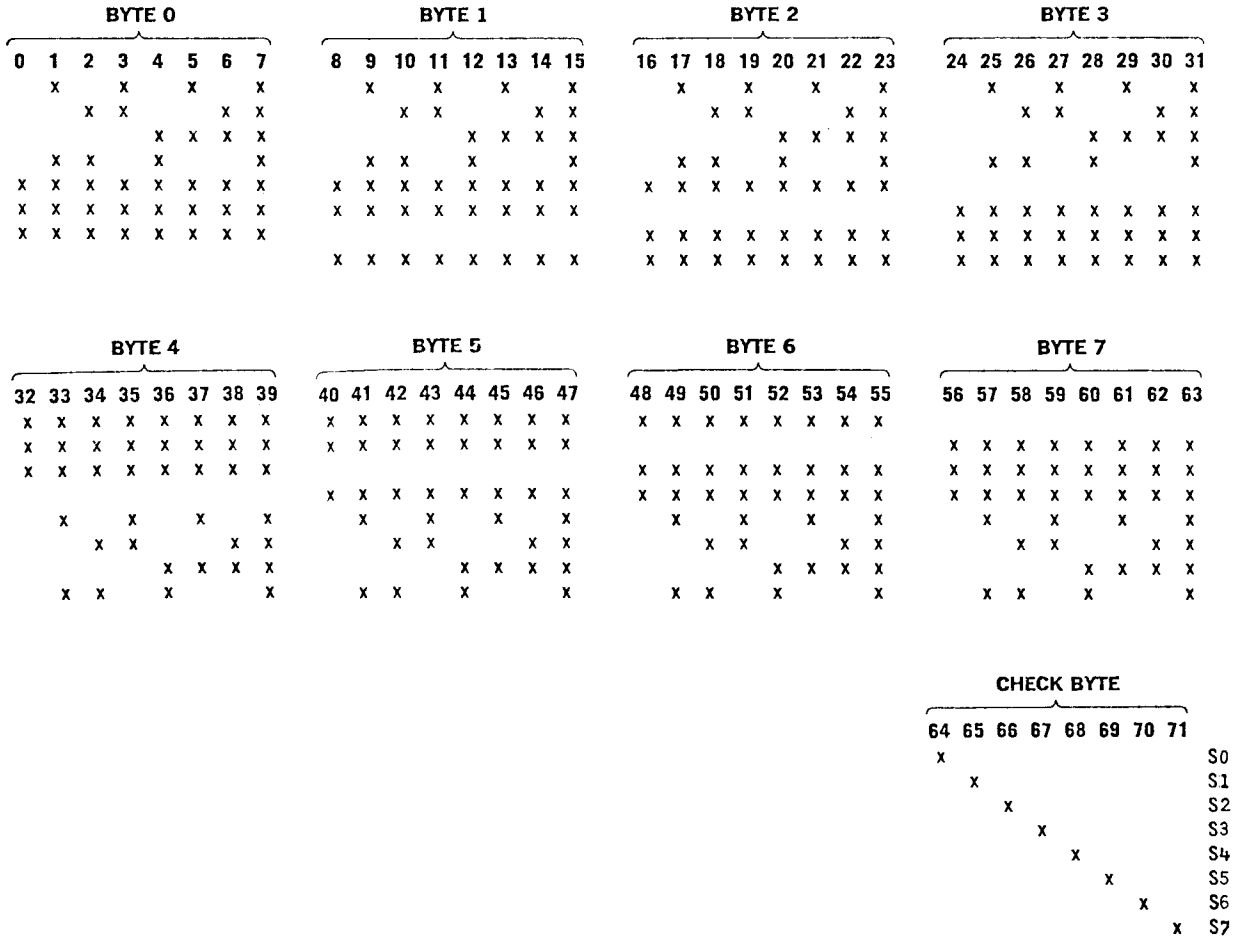


Figure 5-4. Error correction matrix

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 SEC-DED decodes the syndrome bits and determines the error condition using the following four rules:

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 multiple errors 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.

SECTION 6

INPUT/OUTPUT SECTION

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

Each input channel consists of a data channel (16 data bits and 3 control bits), a 64-bit assembly register, a current address register, and a limit register. Each input channel can cause a CPU interrupt condition when the current address equals the limit register value or when the input device sends a disconnect.

Each output channel consists of a data channel (16 data bits and 3 control bits), a 64-bit disassembly register, a current address register, and a limit register. Each output channel can cause a CPU interrupt condition when the current address equals the limit register value. A disconnect is sent on the output channel after the last word of a record is sent and acknowledged.

MEMORY ACCESS

Each of the four channel groups is assigned a time slot, 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 that is accepted 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 had a memory request denied 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.

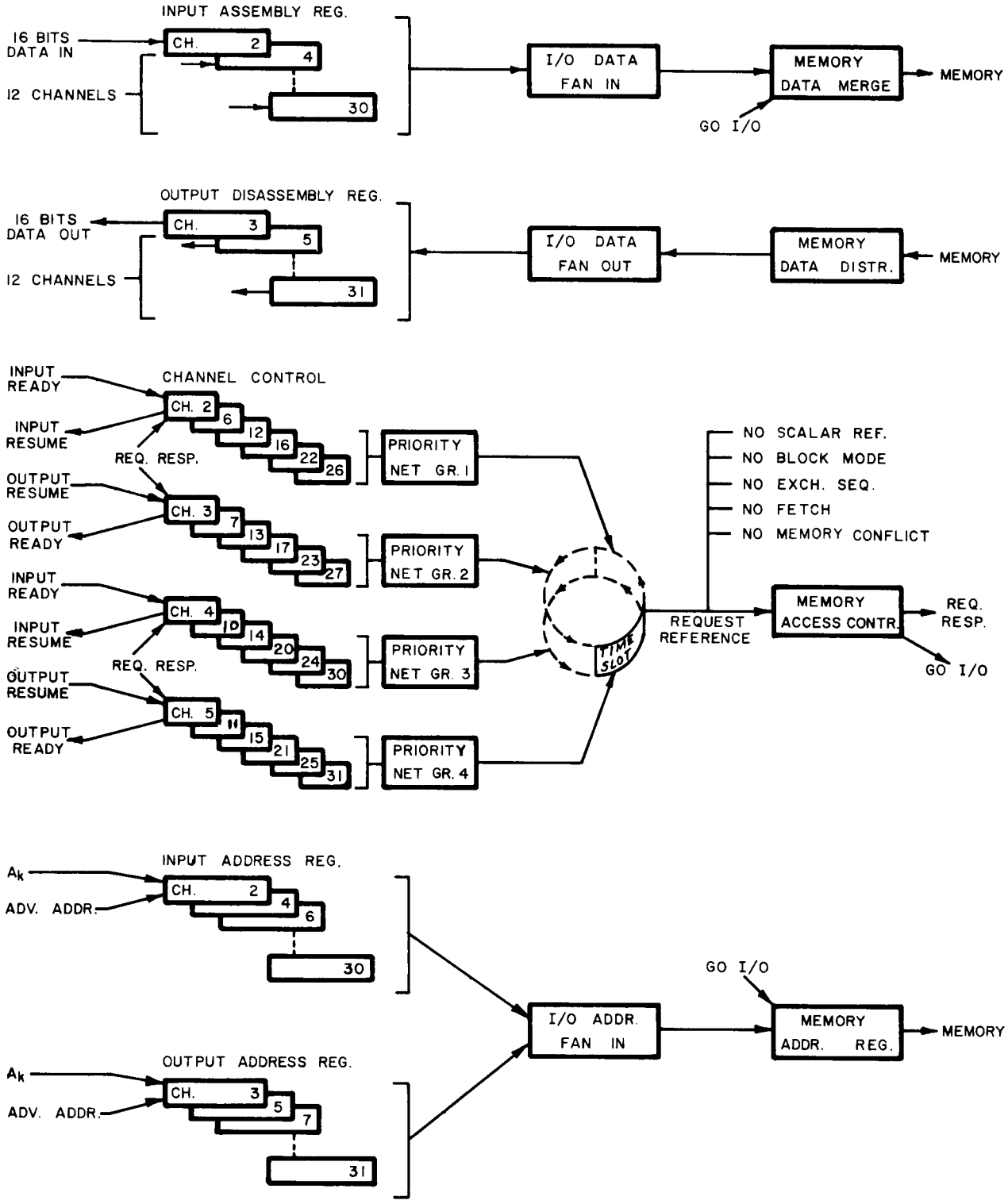


Figure 6-1. Channel I/O control

Maximum request rates are the following:

1 channel	8 CPs
1 channel group	4 CPs
All channel groups	1 CP

I/O CHANNEL PARITY

Channel parity is maintained by one parity bit for each four data bits on the 16-bit channels.

On input, the CRAY-1 samples parity with the data and, if it senses a parity error, immediately generates a channel error interrupt.

On output, the parity bits are sent with the data and are available for checking by the receiving device.

RESYNCHRONIZATION

Each channel resynchronizes the control pulses received from the external devices. Control pulses must be a minimum of 50 nsec ± 10 nsec. The maximum rate is one pulse each 100 nsec.

MEMORY BANK CONFLICTS

Memory bank conflicts are tested for CPU scalar references and I/O memory references. All other memory references (block transfers, exchange sequence, instruction fetch sequence) wait issue until all memory banks are quiet.

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 four 1-clock-period registers. The first register is rank A, the second register is rank B, the third register is rank C, and the fourth register is the memory address register.

Rank A is not explicitly tested against on a scalar reference since I/O is prohibited for scalar in CP2. Scalar causes a three clock period hold of the new request. The lower four bits† of a scalar reference address are tested against ranks B and C. Coincidence with rank B causes a two clock

† 3 bits for 8-bank phasing; see description in section 5.

period hold of the new request; coincidence with rank C causes a one clock period hold of the new request. A lack of coincidence allows a new memory reference to proceed without delay.

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. An I/O request that is disallowed must wait eight clock periods before it can request again.

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 in process
6. No exchange sequence
7. No 033 request

I/O LOCKOUT

An I/O memory request can be locked out by a block transfer. Multiple block transfers cannot issue without allowing waiting I/O references 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.

I/O INTERRUPTS

I/O interrupts can be caused by the following:

1. Current address equals limit address
2. External device disconnect on an input channel
3. Channel error condition

† 3 bits for 8-bank phasing; see description in section 5.

The channel number causing an interrupt can be determined by use of a 033 instruction which will read to Ai the highest priority channel number requesting an interrupt. The lowest numbered channel has the highest priority. The interrupt request will continue until cleared by the monitor program.

CHANNEL ERROR CONDITIONS

A channel error condition can be generated as follows:

1. Input channel not active and input data ready
2. Input channel current address equals limit address and input ready
3. Input ready received before channel has written previous data in memory (extra ready)
4. Output channel not active and output resume
5. Parity error on an input channel

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.

INPUT CHANNEL SIGNALS

INPUT DATA

There are sixteen lines for input data. Data must arrive coincident with the input ready signal and remain until an input resume signal is sent to the sending device.

INPUT READY

This signal indicates that data is on the input lines. The input ready signal is resynchronized before sampling the input data. This signal must be a pulse with a minimum width of 50 nsec ± 10 nsec.

INPUT RESUME

This signal indicates that the input data has been received. The input data lines can now receive a new data word. This signal is a 50 nsec pulse.

INPUT DISCONNECT

This signal terminates an input sequence and causes the input channel to interrupt the CPU. This signal must be a pulse with a minimum width of 50 nsec ± 10 nsec.

OUTPUT CHANNEL SIGNALS

OUTPUT DATA

There are sixteen lines for output data. Data is sent coincident with the output ready signal and remains on the line until an output resume signal is received.

OUTPUT READY

This signal indicates that data is on the output lines. This signal is a 50 nsec pulse.

OUTPUT RESUME

This signal indicates the receiving device has received the output data. The output resume is resynchronized allowing a new output data word to change the output data lines. This signal must be a pulse with a minimum width of 50 nsec ± 10 nsec.

OUTPUT DISCONNECT

This signal terminates an output sequence when the last data word in a record (current address equals limit address) has been acknowledged. This signal is a 50 nsec pulse.

CHANNEL OPERATION CONTROL

Channel input and output is controlled by the 20-bit channel address (CA) and channel limit (CL) registers. A set of these registers exists for each channel. The CA register contains the address of the next channel

word. The CL register specifies the limit address. In programming the channel, the CL register is first initialized with the channel limit address and then the channel is activated by setting the beginning address in CA. During the transfer, (CA) increments toward (CL). When (CA) equals (CL), the transfer is complete and an interrupt occurs. The last word read or written is at (CL)-1. The monitor may then determine which channel caused the interrupt and take appropriate action.

Any channel may read or write elsewhere in memory with any size block but does so only under the exclusive control of the monitor program.

I/O CHANNEL MASTER CLEAR

The following program sequence master clears an I/O channel. The sequence affects the specified channel only.

0011jk	Set limit address	(LA)
0010jk	Set current address	(CA)=(LA)
0012jk	Clear interrupt	Enables MC
	(Delay, device dependent)	
0011jk	Set limit address	(LA); disables MC

The device type determines the length of time to wait after turning off MC before trying to program it.

Only one channel of an I/O pair responds to this sequence.

For a normal channel pair, program the input channel and set the delay at 1 μsec.

For a high-speed synchronous channel pair, program the output channel and set the delay at one clock period.

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.

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 only be reset by the 0014 monitor instruction.

APPENDIX SECTION

SUMMARY OF TIMING INFORMATION A

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 scalar mask instruction 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. An asterisk indicates that issue may be delayed because of a functional unit reservation by a vector instruction. Memory may be considered a functional unit for timing considerations.

(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 referemces 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.

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.

† Multiple of 4 for 8-bank phasing; refer to section 5.

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.

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

† 18 clock periods for 8-bank phasing option; refer to section 5.

HOLD ISSUE

A delay of issue results if a 100 - 137 instruction is in the CIP 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 CIP 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 conflict is sensed.

MODULE TYPES

Alpha Code	Application	No. Used
	A SERIES MODULES	
AA	Address adder	5
AB	Storage block address	2
AC	Vector storage control	1
AD	Storage address distribution	3
AE	B and T storage control	1
AF	Address multiply levels 1 and 2	3
AG	Address multiply level 2	3
AH	Address multiply upper level 3	1
AI	Address multiply lower level 3	1
AJ	Address multiply level 4	1
AR	Address registers	12
	D SERIES MODULES	
DA*	Input channel control 8-bit	12
DB*	Output channel control 8-bit	12
DC*	Input data assembly 8-bit	12
DD*	Output data disassembly 8-bit	12
DE	Address merge fanout	10
DF	Channel reference control	1
DG	Channel interrupt control	1
DH	Channel address control	1
DI	Synchronizing circuits	3
DJ	Input channel control 16-bit	12
DK	Output channel control 16-bit	12
DL	Input data assembly 16-bit	12
DM	Output data disassembly 16-bit	12
	F SERIES MODULES	
FA	Floating add exponent input operands	1
FB	Floating add exponent input operands	1
FC	Floating add coefficient input operands	4
FD	Floating add coefficient alignment	4
FE	Floating add coefficient add (front half)	3
FF	Floating add coefficient add (back half)	3
FG	Floating add coefficient result	2
FH	Floating add coefficient result	1
FI	Floating add exponent data	1
FJ	Floating add exponent result	1
	G SERIES MODULES	
GA	Scalar single shift	4
GB	Scalar double shift (front half)	4
GC	Scalar double shift (back half)	4
GD	Data Ak to Si extended	1
GE	Scalar add (front half)	4
GF	Scalar add (back half)	2
GG	Constant to Si	1
GH	Pop and zero count to Ai	1
GI	Real time clock	2
GR	Scalar registers	32
	H SERIES MODULES	
HA	Program branch control	1
HB	Next instruction parcel	4
HC	Lower program address	1
HD	Upper program address	2
HE	Program parameter data	4
HF	Fetch sequence control	1
HR	Instruction buffers	8
HX	Exchange sequence control	1
	J SERIES MODULES	
JA	CIP fanout to AR modules	5
JB	CIP fanout to GR modules	10
JC	Select vector data paths	1
JD	Vector function issue control	1
JE	Floating point issue control	1
JF	Vector register issue control	1
JG	Scalar register issue control	1
JH	Address register issue control	1
JI	Storage access issue control	1
JJ	Hold storage issue control	1
JK	Address access control	1
JL	Scalar access control	1
	M SERIES MODULES	
MA	First level product	24
MB	Second level product	10
MC	Third level product	8
MD	Fourth level product	3
ME	Fifth level product	3
MF	First level ends	2
MG	First section exponents	1
MH	Last section exponents	1

* 16-bit channels can be substituted for 8-bit channels in groups of four

Alpha Code	Application	No. Used
	R SERIES MODULES	
RA	Table for Ao	1
RB	Table for Ao^2	1
RC	Form A_1	3
RD	Form A_1	1
RE	Form A_1	1
RF	Form A_1	2
RG	Form A_1	1
RH	Form A_1^2	2
RI	Form $A_{1_2}^2$	1
RJ	Form $A_{1_2}^2$	1
RK	Form $A_{1_2}^2$	2
RL	Form A_1^2	1
RM	Form A_2	10
RN	Form A_2	2
RO	Form A_2	7
RP	Form A_2	1
RQ	Form A_2	3
RR	Form A_2	1
RS	Reciprocal coefficient	2
RT	Reciprocal coefficient	2
RU	Operand delay	9
RV	Result exponent	1
	S SERIES MODULES	
SH*	16-bit synchronous input data assembly	
SI*	16-bit synchronous output data assembly	

Alpha Code	Application	No. Used
	T SERIES MODULES	
TC	Clock fanout	9
TX**	16-bank phasing	2
TY**	8-bank phasing	2
TZ	Master clock	1
	V SERIES MODULES	
VA	Data to vector registers	32
VB	Vector data to jk functions	32
VC	Vector data to j functions	16
VD	Vector length control	1
VE	Vector write control	1
VF	Front half vector shift	4
VG	Back half vector shift	4
VH	Front half vector add	4
VI	Back half vector add	2
VJ	Vector logical data	4
VK	Vector logical control	1
VR	Vector registers	32
	Z SERIES MODULES***	
ZB	Storage with memory data buffers	288
ZC	Storage with clock fanout	36
ZD	Storage R/W control	1
ZE	Storage section control	2
ZF	Storage with address fanout	120
ZG	Check bit generation	2
ZI	Correction storage	1
ZK	Syndrome generation and error correction	32
ZZ	Storage module	708

* SH and SI modules can be substituted for DJ and DK modules, respectively, and are used for the disk controller.

** For 8-bank phasing, TY modules are substituted for TX modules.

*** Figures are for 16-bank memory.

SOFTWARE CONSIDERATIONS C

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. 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.

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)
††0014jx	RT	Sj	4-8	-	Enter real-time clock register with (Sj)
0020xk	VL	Ak	4-10	-	Transmit (Ak) to VL register
†0020x0	VL	1	4-10	-	Transmit 1 to VL register
0021xx	EFI		4-10.1	-	Enable interrupt on floating point error
0022xx	DFI		4-10.1	-	Disable interrupt on floating point error
003xjx	VM	Sj	4-11	-	Transmit (Sj) to VM register
†003x0x	VM	0	4-11	-	Clear VM register
004xxx	EX		4-12	-	Normal exit
†004ijk	EX	exp	4-12	-	Normal exit
005xjk	J	Bjk	4-13	-	Jump to (Bjk)
006ijkm	J	exp	4-14	-	Jump to exp
007ijkm	R	exp	4-15	-	Return jump to exp; set B00 to P
010ijkm	JAZ	exp	4-16	-	Branch to exp if (A0) = 0
011ijkm	JAN	exp	4-16	-	Branch to exp if (A0) ≠ 0
012ijkm	JAP	exp	4-16	-	Branch to exp if (A0) positive
013ijkm	JAM	exp	4-16	-	Branch to exp if (A0) negative
014ijkm	JSZ	exp	4-17	-	Branch to exp if (S0) = 0
015ijkm	JSN	exp	4-17	-	Branch to exp if (S0) ≠ 0
016ijkm	JSP	exp	4-17	-	Branch to exp if (S0) positive
017ijkm	JSM	exp	4-17	-	Branch to exp if (S0) negative
020ijkm	Ai	exp	4-18	-	Transmit exp = jkm to Ai
021ijkm	Ai	exp	4-18	-	Transmit exp = 1's complement of jkm to Ai
022ijk	Ai	exp	4-19	-	Transmit exp = jk to Ai
023ijx	Ai	Sj	4-20	-	Transmit (Sj) to Ai
024ijk	Ai	Bjk	4-21	-	Transmit (Bjk) to Ai
025ijk	Bjk	Ai	4-21	-	Transmit (Ai) to Bjk
026ijx	Ai	PSj	4-22	Pop/LZ	Population count of (Sj) to Ai
027ijx	Ai	ZSj	4-23	Pop/LZ	Leading zero count of (Sj) to Ai
030ijk	Ai	Aj+Ak	4-24	A Int Add	Integer sum of (Aj) and (Ak) to Ai
†030i0k	Ai	Ak	4-24	A Int Add	Transmit (Ak) to Ai
†030ij0	Ai	Aj+1	4-24	A Int Add	Integer sum of (Aj) and 1 to Ai
031ijk	Ai	Aj-Ak	4-24	A Int Add	Integer difference of (Aj) less (Ak) to Ai
†031i00	Ai	-1	4-24	A Int Add	Transmit -1 to Ai
031i0k	Ai	-Ak	4-24	A Int Add	Transmit the negative of (Ak) to Ai
031ij0	Ai	Aj-1	4-24	A Int Add	Integer difference of (Aj) less 1 to Ai
032ijk	Ai	Aj*Ak	4-25	A Int Mult	Integer product of (Aj) and (Ak) to Ai

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
033i0x	Ai	CI	4-26	-	Channel number to Ai (j=0)
033ij0	Ai	CA,Aj	4-26	-	Address of channel (Aj) to Ai (j≠0; k=0)
033ij1	Ai	CE,Aj	4-26	-	Error flag of channel (Aj) to Ai (j≠0; k=1)
034ijk	Bjk,Ai	,A0	4-28	Memory	Read (Ai) words to B register jk from (A0)
†034ijk	Bjk,Ai	0,A0	4-28	Memory	Read (Ai) words to B register jk from (A0)
035ijk	,A0	Bjk,Ai	4-28	Memory	Store (Ai) words at B register jk to (A0)
†035ijk	0,A0	Bjk,Ai	4-28	Memory	Store (Ai) words at B register jk to (A0)
036ijk	Tjk,Ai	,A0	4-28	Memory	Read (Ai) words to T register jk from (A0)
†036ijk	Tjk,Ai	0,A0	4-28	Memory	Read (Ai) words to T register jk from (A0)
037ijk	,A0	Tjk,Ai	4-28	Memory	Store (Ai) words at T register jk to (A0)
†037ijk	0,A0	Tjk,Ai	4-28	Memory	Store (Ai) words at T register jk to (A0)
040ijkm	Si	exp	4-30	-	Transmit jkm to Si
041ijkm	Si	exp	4-30	-	Transmit exp = 1's complement of jkm to Si
042ijk	Si Si	<exp #>exp	4-31	S Logical	Form 1's mask exp = 64-jk bits in Si from the right
†042i77	Si	1	4-31	S Logical	Enter 1 into Si
†042i00	Si	-1	4-31	S Logical	Enter -1 into Si
043ijk	Si Si	>exp #<exp	4-31	S Logical	Form 1's mask exp = jk bits in Si from the left
†043i00	Si	0	4-31	S Logical	Clear Si
044ijk	Si	Sj&Sk	4-33	S Logical	Logical product of (Sj) and (Sk) to Si
†044ij0	Si	Sj&SB	4-33	S Logical	Sign bit of (Sj) to Si
†044ij0	Si	SB&Sj	4-33	S Logical	Sign bit of (Sj) to Si (j≠0)
†045ijk	Si	#Sk&Sj	4-33	S Logical	Logical product of (Sj) and 1's complement of (Sk) to Si
†045ij0	Si	#SB&Sj	4-33	S Logical	(Sj) with sign bit cleared to Si
046ijk	Si	Sj\Sk	4-33	S Logical	Logical difference of (Sj) and (Sk) to Si
†046ij0	Si	Sj\SB	4-33	S Logical	Toggle sign bit of Sj, then enter into Si
†046ij0	Si	SB\Sj	4-33	S Logical	Toggle sign bit of Sj, then enter into Si (j≠0)
047ijk	Si	#Sj\Sk	4-33	S Logical	Logical equivalence of (Sk) and (Sj) to Si
†047i0k	Si	#Sk	4-33	S Logical	Transmit 1's complement of (Sk) to Si
†047ij0	Si	#Sj\SB	4-33	S Logical	Logical equivalence of (Sj) and sign bit to Si
†047ij0	Si	#SB\Sj	4-33	S Logical	Logical equivalence of (Sj) and sign bit to Si (j≠0)
†047i00	Si	#SB	4-33	S Logical	Enter 1's complement of sign bit into Si
050ijk	Si	Sj!Si&Sk	4-33	S Logical	Logical product of (Si) and (Sk) complement ORed with logical product of (Sj) and (Sk) to Si
†050ij0	Si	Sj!Si&SB	4-33	S Logical	Scalar merge of (Si) and sign bit of (Sj) to Si
051ijk	Si	Sj!Sk	4-33	S Logical	Logical sum of (Sj) and (Sk) to Si
†051i0k	Si	Sk	4-33	S Logical	Transmit (Sk) to Si
†051ij0	Si	Sj!SB	4-33	S Logical	Logical sum of (Sj) and sign bit to Si
†051ij0	Si	SB!Sj	4-33	S Logical	Logical sum of (Sj) and sign bit to Si (j≠0)
†051i00	Si	SB	4-33	S Logical	Enter sign bit into Si
052ijk	S0	Si<exp	4-35	S Shift	Shift (Si) left exp = jk places to S0
053ijk	S0	Si>exp	4-35	S Shift	Shift (Si) right exp = 64-jk places to S0
054ijk	Si	Si<exp	4-35	S Shift	Shift (Si) left exp = jk places
055ijk	Si	Si>exp	4-35	S Shift	Shift (Si) right exp = 64-jk places
056ijk	Si	Si,Sj<Ak	4-36	S Shift	Shift (Si and Sj) left (Ak) places to Si
†056ij0	Si	Si,Sj<1	4-36	S Shift	Shift (Si and Sj) left one place to Si
†056i0k	Si	Si<Ak	4-36	S Shift	Shift (Si) left (Ak) places to Si

† Special syntax form
†† Privileged to monitor mode.

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
057ijk	Si	Sj,Si Ak	4-36	S Shift	Shift (Sj and Si) right (Ak) places to Si
†057ij0	Si	Sj,Si>1	4-36	S Shift	Shift (Sj and Si) right one place to Si
†057i0k	Si	Si>Ak	4-36	S Shift	Shift (Si) right (Ak) places to Si
060ijk	Si	Sj+Sk	4-37	S Int Add	Integer sum of (Sj) and (Sk) to Si
061ijk	Si	Sj-Sk	4-37	S Int Add	Integer difference of (Sj) and (Sk) to Si
†061i0k	Si	-Sk	4-37	S Int Add	Transmit negative of (Sk) to Si
062ijk	Si	Sj+FSk	4-38	F.P. Add	Floating sum of (Sj) and (Sk) to Si
†062i0k	Si	+FSk	4-38	F.P. Add	Normalize (Sk) to Si
063ijk	Si	Sj-FSk	4-38	F.P. Add	Floating difference of (Sj) and (Sk) to Si
†063i0k	Si	-FSk	4-38	F.P. Add	Transmit normalized negative of (Sk) to Si
064ijk	Si	Sj*FSk	4-38	F.P. Mult	Floating product of (Sj) and (Sk) to Si
065ijk	Si	Sj*HSk	4-39	F.P. Mult	Half precision rounded floating product of (Sj) and (Sk) to Si
066ijk	Si	Sj*RSk	4-39	F.P. Mult	Full precision rounded floating product of (Sj) and (Sk) to Si
067ijk	Si	Sj*ISk	4-39	F.P. Mult	2 - Floating product of (Sj) and (Sk) to Si
070ijx	Si	/HSj	4-41	F.P. Rcpl	Floating reciprocal approximation of (Sj) to Si
071i0k	Si	Ak	4-42	-	Transmit (Ak) to Si with no sign extension
071i1k	Si	+Ak	4-42	-	Transmit (Ak) to Si with sign extension
071i2k	Si	+FAk	4-42	-	Transmit (Ak) to Si as unnormalized floating point number
071i3x	Si	0.6	4-42	-	Transmit constant 0.75*2**48 to Si
071i4x	Si	0.4	4-42	-	Transmit constant 0.5 to Si
071i5x	Si	1.	4-42	-	Transmit constant 1.0 to Si
071i6x	Si	2.	4-42	-	Transmit constant 2.0 to Si
071i7x	Si	4.	4-42	-	Transmit constant 4.0 to Si
072ixx	Si	RT	4-44	-	Transmit (RTC) to Si
073ixx	Si	VM	4-44	-	Transmit (VM) to Si
074ijk	Si	Tjk	4-44	-	Transmit (Tjk) to Si
075ijk	Tjk	Si	4-44	-	Transmit (Si) to Tjk
076ijk	Si	Vj,Ak	4-45	-	Transmit (Vj, element (Ak)) to Si
077ijk	Vi,Ak	Sj	4-45	-	Transmit (Sj) to Vi element (Ak)
†077i0k	Vi,Ak	0	4-45	-	Clear Vi element (Ak)
10hijkm	Ai	exp,Ah	4-46	Memory	Read from ((Ah) + exp) to Ai (A0=0)
†100ijkm	Ai	exp,0	4-46	Memory	Read from (exp) to Ai
†100ijkm	Ai	exp,	4-46	Memory	Read from (exp) to Ai
†10hi000	Ai	,Ah	4-46	Memory	Read from (Ah) to Ai
11hijkm	exp,Ah	Ai	4-46	Memory	Store (Ai) to (Ah) + exp (A0=0)
†110ijkm	exp,0	Ai	4-46	Memory	Store (Ai) to exp
†110ijkm	exp,	Ai	4-46	Memory	Store (Ai) to exp
†11hi000	,Ah	Ai	4-46	Memory	Store (Ai) to (Ah)
12hijkm	Si	exp,Ah	4-46	Memory	Read from ((Ah) + exp) to Si (A0=0)
†120ijkm	Si	exp,0	4-46	Memory	Read from exp to Si
†120ijkm	Si	exp,	4-46	Memory	Read from exp to Si
†12hi000	Si	,Ah	4-46	Memory	Read from (Ah) to Si
13hijkm	exp,Ah	Si	4-46	Memory	Store (Si) to (Ah) + exp (A0=0)
†130ijkm	exp,0	Si	4-46	Memory	Store (Si) to exp
†130ijkm	exp,	Si	4-46	Memory	Store (Si) to exp
†13hi000	,Ah	Si	4-46	Memory	Store (Si) to (Ah)
140ijk	Vi	Sj&Vk	4-48	V Logical	Logical products of (Sj) and (Vk) to Vi
†140i00	Vi	0	4-48	V Logical	Clear Vi
141ijk	Vi	Vj&Vk	4-48	V Logical	Logical products of (Vj) and (Vk) to Vi

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
142ijk	Vi	Sj!Vk	4-48	V Logical	Logical sums of (Sj) and (Vk) to Vi
†142i0k	Vi	Vk	4-48	V Logical	Transmit (Vk) to Vi
143ijk	Vi	Vj!Vk	4-48	V Logical	Logical sums of (Vj) and (Vk) to Vi
144ijk	Vi	Sj\Vk	4-48	V Logical	Logical differences of (Sj) and (Vk) to Vi
145ijk	Vi	Vj\Vk	4-48	V Logical	Logical differences of (Vj) and (Vk) to Vi
146ijk	Vi	Sj!Vk&VM	4-48	V Logical	Transmit (Sj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
†146i0k	Vi	#VM&Vk	4-48	V Logical	Vector merge of (Vk) and 0 to Vi
147ijk	Vi	Vj!Vk&VM	4-48	V Logical	Transmit (Vj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
150ijk	Vi	Vj<Ak	4-52	V Shift	Shift (Vj) left (Ak) places to Vi
†150ij0	Vi	Vj<1	4-52	V Shift	Shift (Vj) left one place to Vi
151ijk	Vi	Vj>Ak	4-52	V Shift	Shift (Vj) right (Ak) places to Vi
†151ij0	Vi	Vj>1	4-52	V Shift	Shift (Vj) right one place to Vi
152ijk	Vi	Vj,Vj<Ak	4-53	V Shift	Double shift (Vj) left (Ak) places to Vi
†152ij0	Vi	Vj,Vj<1	4-53	V Shift	Double shift (Vj) left one place to Vi
153ijk	Vi	Vj,Vj>Ak	4-53	V Shift	Double shift (Vj) right (Ak) places to Vi
153ij0	Vi	Vj,Vj>1	4-53	V Shift	Double shift (Vj) right one place to Vi
154ijk	Vi	Sj+Vk	4-56	V Int Add	Integer sums of (Sj) and (Vk) to Vi
155ijk	Vi	Vj+Vk	4-56	V Int Add	Integer sums of (Vj) and (Vk) to Vi
156ijk	Vi	Sj-Vk	4-56	V Int Add	Integer differences of (Sj) and (Vk) to Vi
†156i0k	Vi	-Vk	4-56	V Int Add	Transmit negative of (Vk) to Vi
157ijk	Vi	Vj-Vk	4-56	V Int Add	Integer differences of (Vj) and (Vk) to Vi
160ijk	Vi	Sj*FVk	4-58	F.P. Mult	Floating products of (Sj) and (Vk) to Vi
161ijk	Vi	Vj*FVk	4-58	F.P. Mult	Floating products of (Vj) and (Vk) to Vi
162ijk	Vi	Sj*HVk	4-58	F.P. Mult	Half precision rounded floating products of (Sj) and (Vk) to Vi
163ijk	Vi	Vj*HVk	4-58	F.P. Mult	Half precision rounded floating products of (Vj) and (Vk) to Vi
164ijk	Vi	Sj*RVk	4-58	F.P. Mult	Rounded floating products of (Sj) and (Vk) to Vi
165ijk	Vi	Vj*RVk	4-58	F.P. Mult	Rounded floating products of (Vj) and (Vk) to Vi
166ijk	Vi	Sj*IVk	4-58	F.P. Mult	2 - floating products of (Sj) and (Vk) to Vi
167ijk	Vi	Vj*IVk	4-58	F.P. Mult	2 - floating products of (Vj) and (Vk) to Vi
170ijk	Vi	Sj+FVk	4-61	F.P. Add	Floating sums of (Sj) and (Vk) to Vi
†170i0k	Vi	+FVk	4-61	F.P. Add	Normalize (Vk) to Vi
171ijk	Vi	Vj+FVk	4-61	F.P. Add	Floating sums of (Vj) and (Vk) to Vi
172ijk	Vi	Sj-FVk	4-61	F.P. Add	Floating differences of (Sj) and (Vk) to Vi
†172i0k	Vi	-FVk	4-61	F.P. Add	Transmit normalized negatives of (Vk) to Vi
173ijk	Vi	Vj-FVk	4-61	F.P. Add	Floating differences of (Vj) and (Vk) to Vi
174ijx	Vi	/HVj	4-63	F.P. Rcpl	Floating reciprocal approximations of (Vj) to Vi
175xj0	VM	Vj,Z	4-65	V Logical	VM=1 where (Vj) = 0
175xj1	VM	Vj,N	4-65	V Logical	VM=1 where (Vj) ≠ 0
175xj2	VM	Vj,P	4-65	V Logical	VM=1 where (Vj) positive
175xj3	VM	Vj,M	4-65	V Logical	VM=1 where (Vj) negative
176ixk	Vi	,A0,Ak	4-67	Memory	Read (VL) words to Vi from (A0) incremented by (Ak)
†176ix0	Vi	,A0,1	4-67	Memory	Read (VL) words to Vi from (A0) incremented by 1
177xjk	,A0,Ak	Vj	4-67	Memory	Store (VL) words from Vj to (A0) incremented by (Ak)
†177xj0	,A0,1	Vj	4-67	Memory	Store (VL) words from Vj to (A0) incremented by 1

† Special syntax form

Comment Sheet

Publication Number: 2240004 C
Title: CRAY-1 Hardware Reference Manual

Please feel free to share with us your comments, criticisms, or compliments regarding this publication. We value your feedback. Thank you.

Comments:

Mail to: Publications
CRAY RESEARCH, INC.
7850 Metro Parkway
Suite 213
Minneapolis, MN 55420

CRAY RESEARCH, INC.

CRAY RESEARCH, INC.

Publications Price List

2240000	CAL Assembler Version 1 Reference Manual	$ 5.00
2240001	CRAY-1 Site Planning Reference Manual	3.00
2240002	An Introduction to Vector Processing	1.00
2240003	Instruction Card	nc
2240004	Reference Manual	5.00
2240006	Data General Station Operator's Guide	5.00
2240008	Introduction to the CRAY-1 (Brochure)	1.00
2240009	CRAY-1 FORTRAN (CFT) Reference Manual	8.00
2240011	External Reference Specification, CRAY OS Version 1.0	5.00
2240012	System Programmer's Handbook	20.00
2240013	UPDATE Reference Manual	3.00
2240014	CFT Mathematical Subprogram Library Reference Manual	3.00
2240630	Mass Storage Subsystem Reference Manual	3.00
2240640	MCU Basic Field Engineer's Manual	(internal)

CRAY RESEARCH, INC.

HEADQUARTERS • 7850 Metro Parkway, Suite 213, Minneapolis, MN 55420 • (612) 854-7472
DEVELOPMENT LABORATORY • P.O. Box 169, Chippewa Falls, WI 54729 • (715) 723-0266

CRAY RESEARCH, INC.

CRAY-1® AND CRAY X-MP COMPUTER SYSTEMS

SOLID-STATE STORAGE DEVICE (SSD) REFERENCE MANUAL

HR-0031

CRAY RESEARCH, INC.

CRAY-1® AND CRAY X-MP COMPUTER SYSTEMS

SOLID-STATE STORAGE DEVICE (SSD) REFERENCE MANUAL

HR-0031

Copyright© 1982 by CRAY RESEARCH, INC. This manual or parts thereof may not be reproduced in any form without permission of CRAY RESEARCH, INC.

RECORD OF REVISION

CRAY RESEARCH, INC.

PUBLICATION NUMBER HR-0031

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	December, 1982 - Original printing.

PREFACE

This publication describes the Cray Research, Inc., Solid-state Storage Device (SSD) for Cray Computer Systems. This publication contains operation and programming information for the SSD and assumes the reader has a familiarity with digital computers.

Details of Cray Research, Inc., mainframes that interface with the SSD are given in the CRAY X-MP Series Mainframe Reference Manual, publication HR-0032, the CRAY-1 S Series Mainframe Reference Manual, publication HR-0029, and the CRAY-1 M Series Mainframe Reference Manual, publication HR-0064.

//

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

//

CONTENTS

PREFACE iii

1. GENERAL INFORMATION 1-1

INTRODUCTION 1-1
Conventions 1-1
SSD MEMORY PORTS 1-3
Port 0 - Internal Memory Refresh 1-4
Port 1 - Maintenance Port 1-4
Port 2 - CRAY-1 S and CRAY-1 M Port 1-4
Port 3 - CRAY X-MP Port 1-4
DATA PROTECTION 1-5
SSD MEMORY 1-5
Memory control and addressing 1-6
Memory cycle 1-6

2. SSD AND CRAY X-MP OPERATION 2-1

INTRODUCTION 2-1
SSD WRITE OPERATION 2-2
SSD READ OPERATION 2-2
SSD PROGRAMMING 2-3
SSD block address 2-3
Central Memory address 2-3
Transfer block length read/write mode 2-3
Status word error flags 2-4
Block length registers 2-4
Clear interface/abort sequence 2-4
Master clear after power on 2-5

3. SSD AND CRAY-1 S OR CRAY-1 M OPERATION 3-1

INTRODUCTION 3-1
HIGH SPEED CONTROLLER 3-1
HSC COMMAND CHANNEL 3-3
HSC command word 3-4
Read/write field 3-4
CPU loopback field 3-5
Clear 100 Mbytes per second channel error field . . 3-5
Diagnostic mode field 3-6

HSC command word (continued)
Transfer length field 3-6
CPU address field 3-6
SSD address field 3-7
HSC STATUS CHANNEL 3-7
HSC status word 3-8
Source field 3-8
Status code field 3-9
SSD PROGRAMMING 3-13
Channel instructions 3-13
Channel master clear sequence 3-15
HSC programming sequence 3-15
HSC command sequence example 3-15
HSC Status Channel interrupt routine example 3-16

FIGURES

1-1 SSD chassis 1-2
1-2 SSD memory organization 1-6
2-1 CRAY X-MP Series Computer System with SSD 2-1
3-1 CRAY-1 S Series Computer System with SSD 3-2
3-2 Data and control paths between SSD and CRAY-1 S or CRAY-1 M mainframe 3-3
3-3 HSC command word format 3-5
3-4 Address words sent from HSC to CPU side of 100 Mbytes per second channel 3-6
3-5 Address words sent from HSC to SSD port interface 3-7
3-6 HSC status word format 3-8

TABLES

1-1 SSD operating characteristics 1-3
1-2 Memory cycles 1-7
3-1 Command channel error status 3-9
3-2 SSD read status codes 3-9
3-3 SSD write status codes 3-11
3-4 Channel instruction summary 3-14

INDEX

GENERAL DESCRIPTION 1

INTRODUCTION

The Cray Research, Inc., Solid-state Storage Device (SSD) is a device for temporary storage of datasets and can be configured with a CRAY X-MP, a CRAY-1 S, or a CRAY-1 M Computer System to significantly increase data transfer rates. The maximum data transfer burst rate is dependent upon the SSD memory size and configuration.

The SSD is housed in a stand-alone, 4-column chassis (figure 1-1). Three columns contain the SSD memory modules; one column contains channels and control logic. The power supplies and cooling system are similar to those used in a Cray mainframe. The bench at the base of each column houses the DC power supplies for that column, and a power distribution unit for the SSD supplies 400 Hz power from the Cray Computer System motor-generators to the SSD chassis power supplies. The SSD chassis is cooled by the system condensing unit. An auxiliary condensing unit may be required to support increased system heat load.

The SSD is located adjacent to a CRAY X-MP mainframe but can be 70 feet from a CRAY-1 S or a CRAY-1 M mainframe.

Operating characteristics of the SSD are summarized in table 1-1. SSD memory ports, data protection, and memory are described in this section.

CONVENTIONS

Unless otherwise indicated, numbers in this manual are decimal. Octal numbers are indicated with an 8 subscript.

Italicized lowercase letters, such as *jk*, indicate variable information.

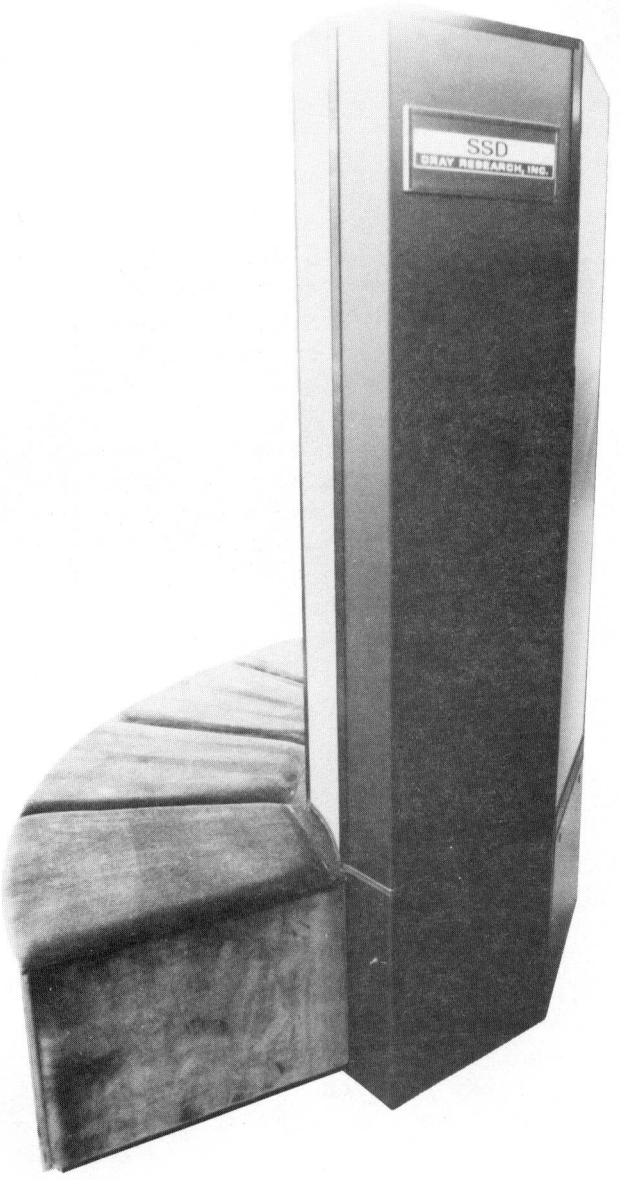


Figure 1-1. Solid-state Storage Device

Table 1-1. SSD operating characteristics

Characteristic	Description
Word size	64 data bits and 8 check bits
Memory size	8,388,608 64-bit words 16,777,216 64-bit words 33,554,432 64-bit words
Number of 64-word blocks	65,536 for 8-million word memory 131,072 for 16-million word memory 262,144 for 32-million word memory
Memory Ports	Port 0 - Internal memory refresh Port 1 - Maintenance port Port 2 - CRAY-1 S or CRAY-1 M mainframe port Port 3 - CRAY X-MP mainframe port
Maximum data transfer burst rate for:	
Port 2	100 Mbytes per second for 8-million word, 16-million word, or 32-million word memory
Port 3	320 Mbytes per second for 8-million word memory 640 Mbytes per second for 16-million word memory 1250 Mbytes per second for 32-million word memory
Data protection	SECDED in memory SECDED on I/O interface

SSD MEMORY PORTS

The SSD has four memory ports. One is used internally; one is used for maintenance purposes; and two are used for data input and output. All four ports connect to SSD memory and are described in the following paragraphs. An Error Log passes port error information and memory error

information to a maintenance computer, and the maintenance computer program logs the error information for maintenance analysis.

PORT 0 - INTERNAL MEMORY REFRESH

Port 0 controls refreshing of SSD memory. Because SSD memory is refreshed within each memory chip, there is no actual data path associated with Port 0. Memory is refreshed every two milliseconds. The refresh operation uses much of the SSD control logic also used in other port operations.

PORT 1 - MAINTENANCE PORT

Port 1, the maintenance port, connects the SSD to a maintenance computer through a 6 Mbytes per second channel that has 16-bit asynchronous control logic.† Port 1 is programmable and executes maintenance commands received from the maintenance computer. Port 1 cannot be used as a data input or data output port.

PORT 2 - CRAY-1 S OR CRAY-1 M PORT

Port 2 connects the SSD with a CRAY-1 S or a CRAY-1 M mainframe by adapting the SSD to a Memory Channel that has a maximum data transfer rate of 100 Mbytes per second.†† The channel pair is connected to a CRAY-1 S or CRAY-1 M mainframe and has an input (write to SSD) and an output (read from SSD) channel. Section 3 of this manual describes the SSD and CRAY-1 S or CRAY-1 M mainframe operation.

PORT 3 - CRAY X-MP PORT

Port 3 connects the SSD directly to the CRAY X-MP mainframe using a channel that has maximum data transfer rate of 1250 Mbytes per second. Section 2 of this manual describes the SSD and CRAY X-MP operation.

---

† For a description of the 6 Mbytes per second (16-bit asynchronous) control channel, refer to a Cray mainframe reference manual.

†† For a description of the 100 Mbytes per second channel, refer to a Cray mainframe reference manual.

DATA PROTECTION

To protect data, single error correction/double error detection (SECDED) logic is used in SSD memory and on data channels to or from the SSD. SECDED logic used is similar to the SECDED logic used in a Cray mainframe Central Memory†.

When data is written into SSD memory, a checkword is generated for the word to be checked and stored with that word. The checkword is an 8-bit Hamming†† code. When the word is read from SSD memory, the checkword and data word are processed to determine if any bits were altered. If no errors occurred, the word is passed to the mainframe.

If an error occurred, the 8 bits of the checkword are analyzed by the logic to find the number of altered bits. If only a single bit was altered, the correction logic resets that bit to the correct state and passes the corrected word out to the mainframe. The Error Log receives details of the error.

If more than a single bit was altered, the logic cannot correct the word. When a double error is detected, it is reported in the 1-word status that is transmitted to the Status Channel (CRAY-1 S or CRAY-1 M configuration) or an error flag is set in the A register (CRAY X-MP configuration). The Error Log receives details of the error.

If more than two bits are in error, results are unpredictable.

SECDED is also used on the data channel when writing data into the SSD. Errors that occur on the channel or in Port 2 or Port 3 logic are corrected and processed as described above. Therefore, there is data protection for write data before storage and data protection for read data after storage.

SSD MEMORY

The SSD has three memory size options: 8 million, 16 million, and 32 million 64-bit words. SSD memory is organized into two groups, Group 0 and Group 1 (figure 1-2) that are used as separate, parallel memories. Each group has up to four sections each divided into eight

† For a description of Central Memory, refer to a Cray mainframe reference manual.

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

banks. The sections and banks used are determined by the SSD memory size and the data transfer rate. All eight sections (64 banks) are used in an SSD with a 32-million word memory; four sections (32 banks) are used in an SSD with a 16-million word memory; and two sections (16 banks) are used in an SSD with an 8-million word memory.

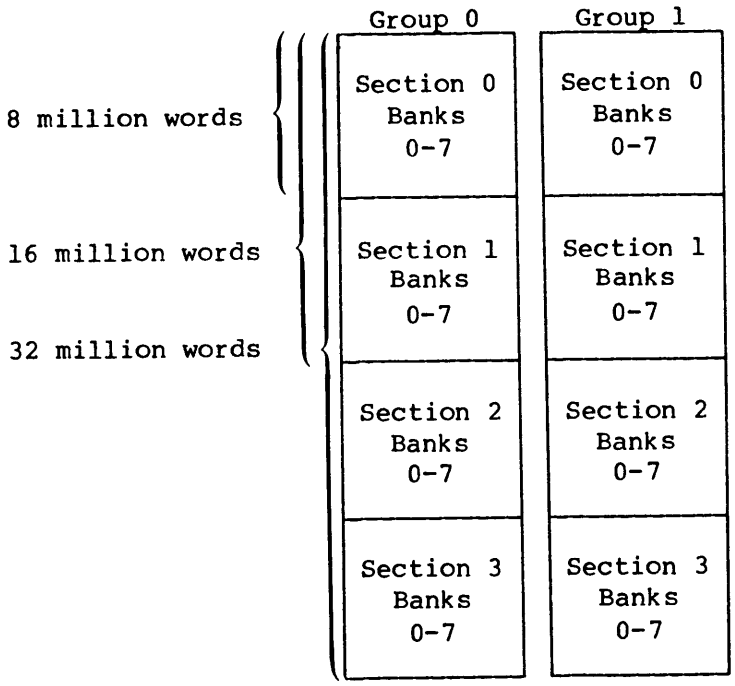


Figure 1-2. SSD memory organization

MEMORY CONTROL AND ADDRESSING

The SSD uses a 64-word block for all transfers into or out of memory. Individual words are not accessible by addressing. A user provides only a starting address for a 64-word block, and a full block is read or written. To read a particular word, the entire block is transferred to Central Memory, then the word is selected using software methods, similar to disk storage data handling methods. Control and addressing are done simultaneously for both memory groups. SSD memory control logic routes each word to the correct location. Once a 64-word block transfer begins, it continues to completion without interruption. Control logic is the same for all memory size options.

MEMORY CYCLE

Control timing of the SSD is based on how long it takes to access half the banks of a 32-million word memory and is the same for all memory size options. After a bank is accessed, there is a delay time (approximately 400 nanoseconds) before it can be accessed again. A bank is considered busy for almost a full memory cycle. Banks are paired as follows: 0 and 4, 1 and 5, 2 and 6, and 3 and 7. If bank n is in the memory cycle, then bank n+4 is also considered busy.

The number of words read or written in one memory cycle depends on the number of sections (size) of SSD memory. Therefore, the number of memory cycles needed to transfer a full 64-word block also depends on the size of memory.

A memory cycle for a 32-million word SSD is the transfer of 32 words from half the total number of memory banks, that is, from 32 of its 64 banks. The first memory cycle uses Banks 0, 1, 2, and 3 of all sections and both groups. The second memory cycle uses Banks 4, 5, 6, and 7 of all sections and both groups.

A 32-million word SSD requires two memory cycles to transfer a 64-word block. A 16-million word and an 8-million word SSD use the same memory cycle time as the 32-million word memory but transfer fewer words in each memory cycle. The 16-million word SSD transfers 16 words per memory cycle with words coming from Group 0 and Group 1, Section 0 and Section 1, Banks 0, 1, 2, and 3 and then Banks 4, 5, 6, and 7. The 8-million word SSD transfers 8 words per memory cycle with words coming from Group 0 and Group 1, Section 0, Banks 0, 1, 2, and 3 and then Banks 4, 5, 6, and 7.

Table 1-2 shows the number of words transferred per memory cycle and number of memory cycles needed to transfer a 64-word block for each SSD memory size.

Table 1-2. Memory cycles

Memory size (words)	Words per memory cycle	Memory cycles per 64-word block
32 million	32	2
16 million	16	4
8 million	8	8

SSD AND CRAY X-MP OPERATION 2

INTRODUCTION

When configuring an SSD with a CRAY X-MP mainframe (figure 2-1), the SSD connects directly to the mainframe using a channel with a burst rate capability of 1250 Mbytes per second.

The SSD uses I/O channels 4, 5, 6, and 7 of the CRAY X-MP.† The SSD provides high speed data transfer to or from Central Memory under the mainframe's software control.

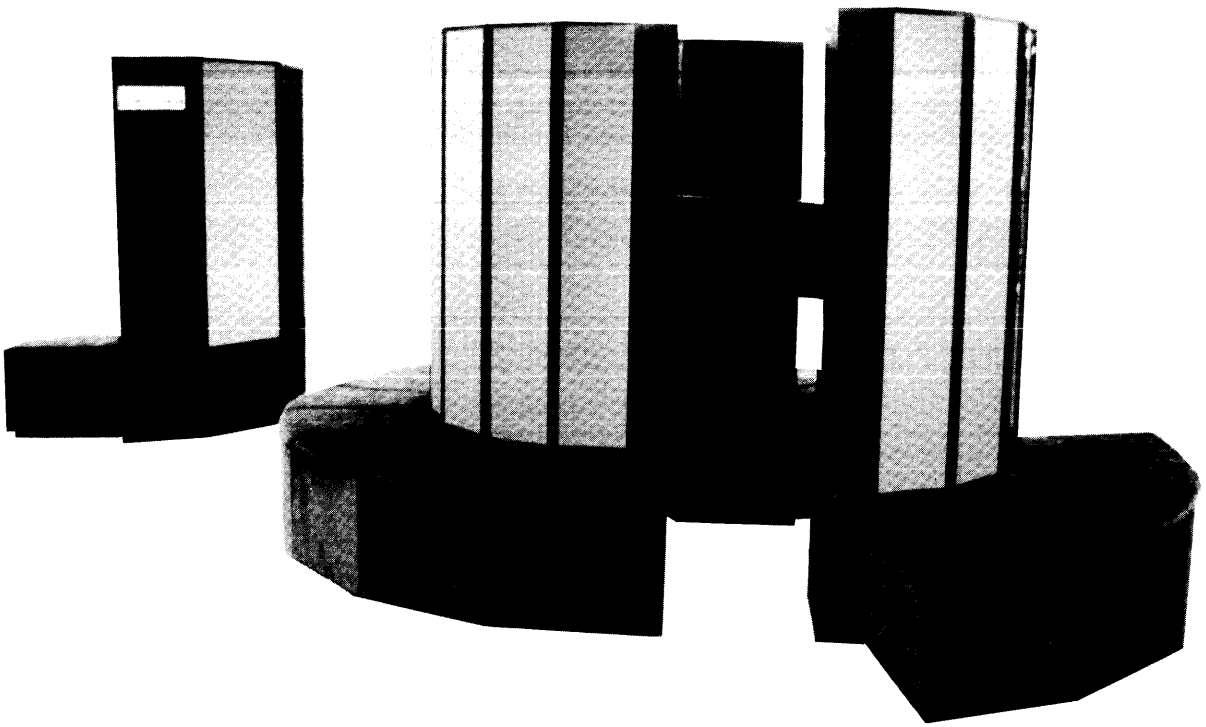


Figure 2-1. CRAY X-MP Series Computer System with SSD

† For additional I/O information, refer to the CRAY X-MP Series Mainframe Reference Manual, publication HR-0032.

Data is transferred between the SSD buffers and the SSD memory at 2 words per SSD clock period (CP). Data is transferred between the SSD buffers and the CRAY X-MP mainframe in 32-word blocks, 2 words per mainframe clock period. For a data transfer, each CPU memory port handles 16 words through one of two buffers. Each CPU memory port handles every other word to or from memory; that is, CPU 0 memory port handles the first CPU address and every other word thereafter. CPU 1 memory port handles every other word starting at CPU address+1.

SSD write and read operations for an SSD operating with a CRAY X-MP and SSD programming information are described in this section.

SSD WRITE OPERATION

Each CPU I/O port has a double buffer containing two 16-word buffers for transmitting data to and receiving data from the SSD. For an SSD write operation, data written to the SSD is first transferred from Central Memory through both CPU I/O ports (one in CPU 0 and one in CPU 1) and written into one of the 16-word buffers of each double buffer. When one 16-word buffer from each double buffer is full, they are paired for a 32-word transfer to the SSD. A check is made to determine if one of the four 32-word buffers in the SSD is available.

When a 32-word buffer is available, the data is sent to that SSD buffer in 16 CPU CPs, two words per CP. While the transfer is in process, the alternate 16-word buffer in each the two CPU I/O port double buffers proceeds to fill in the CPU. When these two 16-word buffers are full, they are paired for a 32-word transfer to the next available 32-word SSD buffer. When a 32-word buffer fills in the SSD, a request is sent to SSD memory, and data is written from the 32-word buffer to SSD memory two words per SSD CP. This operation continues until all data is written to the SSD; then an interrupt is sent to the CPU that initiated the SSD data transfer.

SSD READ OPERATION

For an SSD read operation, when any of the four 32-word buffers in the SSD is empty, a reference is made to read from the SSD memory to Central Memory. Data is then read from SSD memory to a 32-word buffer two words per SSD CP. If the SSD has a full 32-word buffer and each CPU I/O port double buffer has an empty 16-word buffer, the data is sent to the two 16-word buffers at a rate of two words per CPU CP. When that 16-word buffer pair is full, a reference is made from the two CPU I/O memory ports to Central Memory. This operation continues until the last 16-word buffer pair is written to Central Memory, then an interrupt is sent to the CPU that initiated the SSD data transfer.

SSD PROGRAMMING

The 1250 Mbyte channel is available to the software as I/O channel 7 of the CRAY X-MP. A sequence of instructions must be issued to this channel to start the read or write operation. Either a read operation can be active or a write operation can be active. The instruction sequence to start an operation is defined below with (A*j*)=7 for each instruction.

Octal code	CAL syntax	Description
0012*j*0	CI,A*j*	Clear interrupt
0010*jk*	CA,A*j* A*k*	(A*k*)=SSD block address
0010*jk*	CA,A*j* A*k*	(A*k*)=Central Memory address
0011*jk*	CL,A*j* A*k*	(A*k*)=Transfer block length/read write mode; transfer initiated.
------	------	Wait interrupt (channel #7)
033*ij*1	A*i* CE,A*j*	Read error flags
0012*j*0	CI,A*j*	Clear interrupt/clear interface

SSD BLOCK ADDRESS

All transfers to or from the SSD must be in blocks of 64 words. Therefore, the A register value must be the SSD absolute address divided by 100_8.

CENTRAL MEMORY ADDRESS

The Central Memory address is used to transfer information into Central Memory. This address can start anywhere within the address field of the CPU.

TRANSFER BLOCK LENGTH READ/WRITE MODE

The transfer block length is the positive number of 64-word blocks to transfer. The maximum block length is 777777_8. To specify the direction of the transfer, the high-order bit of the A*k* register (instruction 011*jk*) is used. The bit clear indicates a read operation (SSD to Central Memory transfer). The bit set indicates a write operation (Central Memory to SSD transfer).

STATUS WORD ERROR FLAGS

The low-order bits of the A*k* register contain the remaining block count. The high-order bits are used for error flags and status bits as follows.

Bits	Description
2^{23}	Fatal error (bit 2^{22}, 2^{21}, or 2^{20} set)
2^{22}	Central Memory double-bit error
2^{21}	SSD memory double-bit error (during read); double-bit channel error (during write).
2^{20}	Block length error
2^{19}	SSD memory single-bit error (during read); single-bit channel error (during write).
2^{18}	Request in progress
$2^{17}-2^{0}$	Remaining block length

BLOCK LENGTH REGISTERS

Two block length registers connect a port on the SSD to an interface on the CRAY X-MP. One register is on the mainframe side of the interface; one register is on the SSD side. Instruction 033*ij*1 reads only the mainframe side block length register.

NOTE

On an error, the remaining block length counter can be off by up to 5 blocks. This is because there are six 32-word buffers in the path for any one direction, and on some errors it takes time to shut down the channel. Therefore, if an attempt is made to use some of the data read, or to rewrite less than the entire transfer, a possible 5-block discrepancy should be taken into account.

CLEAR INTERFACE/ABORT SEQUENCE

The interface on the CRAY X-MP connects to an SSD port and is cleared after every transfer. This operation takes approximately 15 CPs. Clearing the interface is initiated on the first Clear Interrupt instruction (0012*j*0) after a transfer. The entire interface is cleared, allowing the next transfer to occur. The error flags are cleared in the process. Under normal operating conditions (no errors or

aborts), the next operation can begin immediately following the clear interrupt instruction. A pending operation is held until the clear finishes.

An abort can be attempted by issuing a Clear Interface instruction (0012j0) on a system that is operational. If an abort is attempted or if errors are encountered during the transfer, the Request in Progress status bit (2^{18}) must be checked after the Clear Interrupt instruction (0012j0) and before the new operation executes the Set SSD Address instruction (0010jk). The status bit must be clear before starting the next operation. Because a pending reference to the SSD must complete before a new transfer can start, an abort attempt can vary in time depending on refresh in the SSD and other I/O activity within the SSD. Since the software directly controls the Central Memory address, the address should not be changed before any pending references are completed.

NOTE

The address register on the channels is directly available to instruction 0010jk. Since a new operation can be initiated and stacked when clearing the interface (due to instruction 0012j0), there cannot be any SSD I/O references in progress due to aborts or errors when the stacked instruction 0010jk is issued. The Request in Progress flag (bit 2^{18}) checks for SSD I/O references in progress and should be clear before an instruction 0010jk is issued.

MASTER CLEAR AFTER POWER ON

Clearing the interface on the CRAY X-MP initially clears all controls associated with the SSD port. The following instruction clears the interface:

Octal code	CAL syntax		Description
0012j0	CI,Aj	(Aj)=7	Clear interface

The Request in Progress status bit (2^{18}) should be checked before the first transfer. The Request in Progress bit should never be set for more than 40 CPs.

SSD AND CRAY-1 S OR CRAY-1 M OPERATION

3

INTRODUCTION

The SSD uses the 100 Mbytes per second channel modules on a CRAY-1 S or on a CRAY-1 M mainframe that are otherwise used for transmitting data between an I/O Subsystem and CRAY-1 mainframe. In addition to these modules, the mainframe controls the operations of the channel to the SSD through a special high speed controller (HSC), which is a group of modules installed in the CRAY-1 S or CRAY-1 M mainframe and cabled to both ends of the 100 Mbytes per second channel link. The mainframe sends transfer commands to and receives status information from the HSC using a modified I/O channel pair. This channel pair is referred to as the HSC command channel (output channel) and the HSC status channel (input channel). The HSC sends the appropriate control signals to both ends of the 100 Mbytes per second channel link to control the transfer of commands and status information.

Figure 3-1 shows a CRAY-1 S Series Computer System configured with an SSD. Special use of the the HSC and I/O channel pair and SSD programming information are described in the following paragraphs.

HIGH SPEED CONTROLLER

The high speed controller (HSC) expands the capabilities of a CRAY-1 S or a CRAY-1 M 100 Mbytes per second channel by allowing the central processing unit (CPU) to originate data transfers. Without the HSC, the CPU 100 Mbytes per second channel can only respond to channel requests from other devices (such as the I/O Subsystem).

The HSC is comprised of two logic modules residing in the mainframe. The HSC decodes commands received from the CPU, returns status to the CPU, initiates 100 Mbytes per second channel transfers, detects error conditions, and forces diagnostic conditions. The only HSC connection to the CPU and SSD 100 Mbytes per second channel is via the channel's control cables; the data cables are connected directly between the CPU and SSD. Figure 3-2 shows the data and control paths between a CRAY-1 S or CRAY-1 M mainframe and SSD.

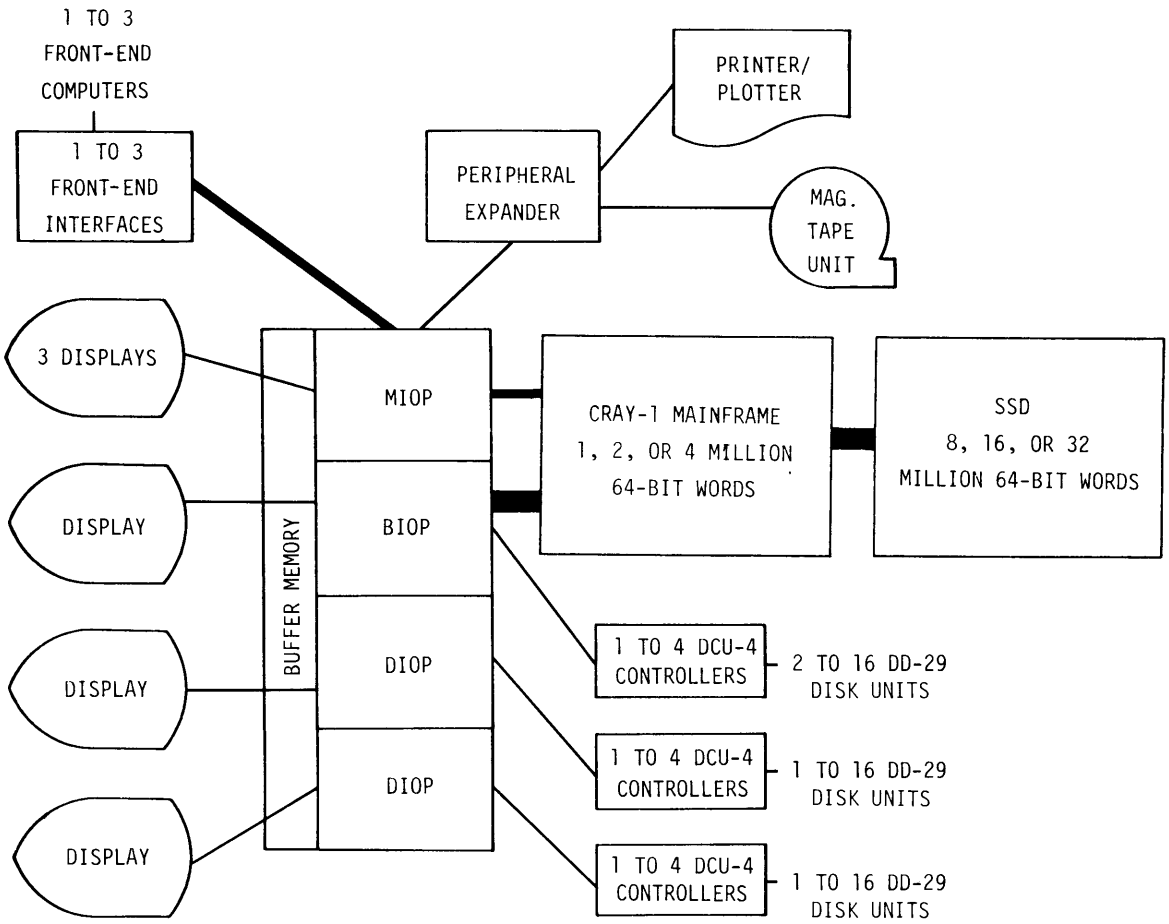


Figure 3-1. CRAY-1 S Series Computer System with SSD

Three control signals (Data Ready, Last Word flag, and Transmit Data) control the flow of data in each direction between a CRAY-1 S or CRAY-1 M mainframe and an SSD. The control signals pass through the HSC unaltered (except under certain diagnostic conditions). The remaining channel signals all originate or terminate at the HSC and are used primarily to initiate data transfers and detect any 100 Mbytes per second channel errors. Once a data transfer is initiated (that is, immediately after the address words are transferred), the data transfer is directed by the three control signals controlling the data flow.

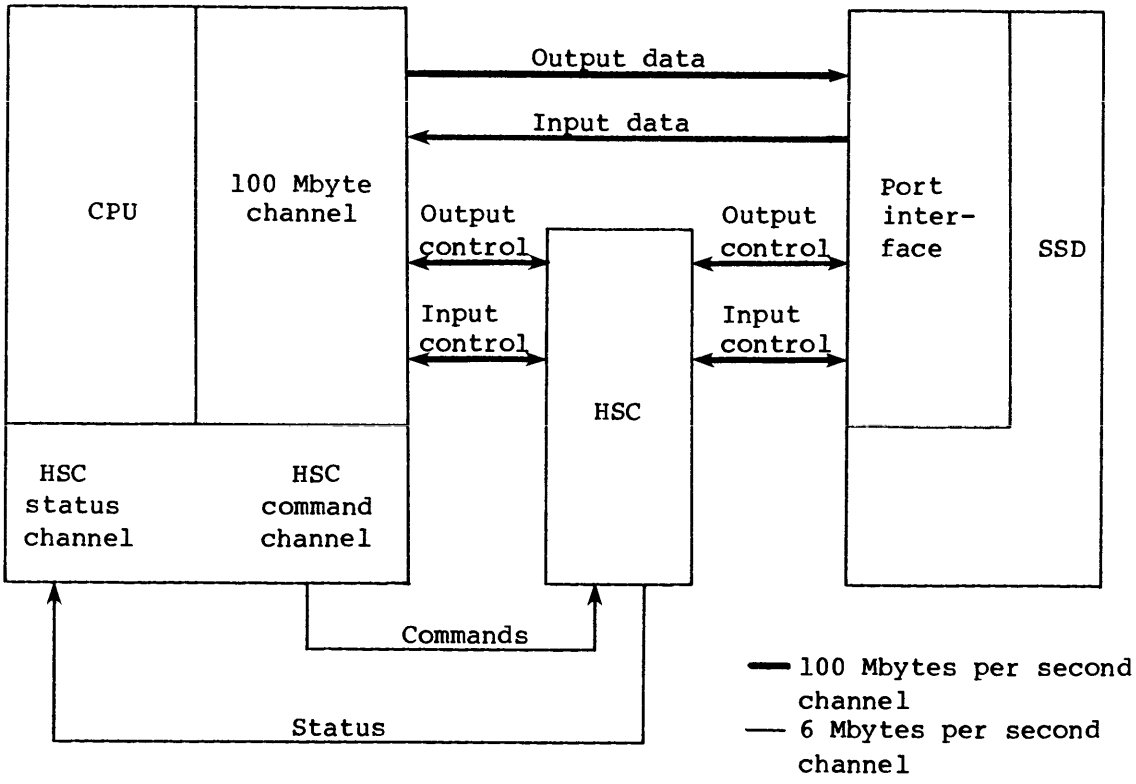


Figure 3-2. Data and control paths between SSD and CRAY-1 S or CRAY-1 M mainframe

HSC COMMAND CHANNEL

To initiate an SSD data transfer, a single 64-bit command word (see subsection on HSC command word) is sent to the HSC from a special mainframe CPU output channel designated the HSC command channel. The HSC command channel occupies one CPU output channel. Because the HSC command channel does not have an interrupt capability, it can be assigned to any output channel without regard to interrupt priority. The HSC command channel differs from a normal CRAY-1 S or CRAY-1 M CPU output channel in the following respects.

Interrupt request: The HSC command channel cannot generate an interrupt request.

Channel Error flag: Since no interrupt capability exists on the HSC command channel,† the function of the Channel Error flag is redefined to allow testing for

† Interrupt capability does exist on the HSC status channel.

Channel Error flag: (continued)

completion of a command transfer. The Channel Error flag is renamed Command Channel Busy flag and is set whenever the HSC command channel is active. The flag remains set until the Command Channel Disconnect is transmitted or until a Clear Channel Interrupt instruction (0012jx) is executed. An I/O master clear from the Maintenance Control Unit, or equivalent, also clears the Command Channel Busy flag. The state of the Command Channel Busy flag is sensed with a normal Read Channel Error flag instruction (033ij1). The usual error information conveyed by the Channel Error flag (that is, resume while inactive or resume during memory reference) does not apply to the HSC command channel.

Channel limit address:

Since all transfers over the HSC command channel consist of a single CPU word, the Channel Limit (CL) register is not implemented. A Set CL instruction (0011jk) for the HSC command channel is executed as a pass. In addition, the Channel Address (CA) register is not incremented after the channel requests a memory reference.

HSC COMMAND WORD

The format of the HSC command word is shown in figure 3-3; the fields are described below.

Read/write field

When set, the 1-bit read/write field indicates that the command pertains to the SSD write data path (write from Central Memory to SSD). When this bit is clear, the command affects the SSD read data path (read to Central Memory from SSD). If the Clear 100 Mbyte Channel Error bit is zero and the diagnostic mode field is not 05, 10, or 15,† an SSD data transfer is initiated. One SSD read operation and one SSD write operation can be in progress simultaneously.

† Mode 05 Transmit Clear Channel pulse to SSD output side of the 100 Mbytes per second channel without deactivating HSC or CPU input side of the channel.

Mode 10 Transmit Last Word flag to CPU input side of the 100 Mbytes per second channel (only if transfer is in progress)

Mode 15 Transmit Clear Channel pulse to CPU input side of the 100 Mbytes per second channel without deactivating HSC or SSD output side of the channel.

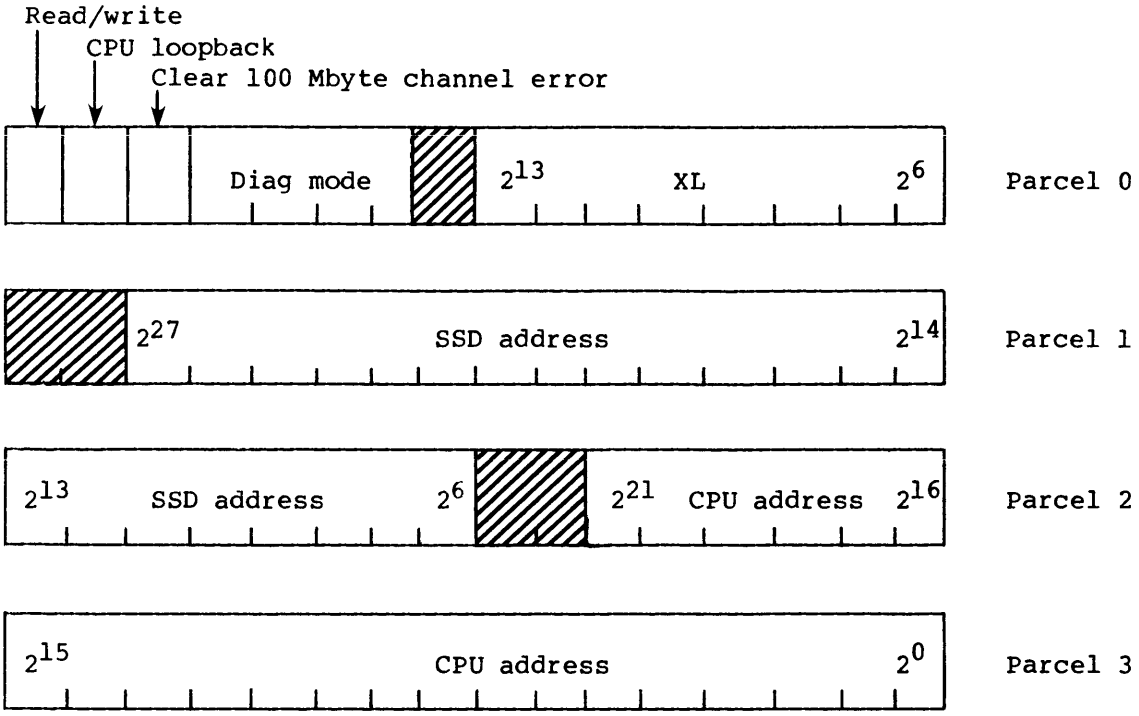


Figure 3-3. HSC command word format

CPU loopback field

The 1-bit CPU loopback field is used for maintenance purposes only. The bit is set only when the CPU output and input sides of the 100 Mbytes per second channel are connected together through the HSC for checking either the HSC or the 100 Mbytes per second channel without using other equipment (that is, the I/O Subsystem or the SSD).

Clear 100 Mbytes per second channel error field

When a 100 Mbytes per second channel error occurs, the channel with the error should be cleared before using it for a subsequent data transfer. Setting this command bit causes a 100-nanosecond Clear Channel signal to be transmitted to the appropriate SSD and CPU sides of the 100 Mbytes per second channel and resets the read or write logic in the HSC. The Read/write bit indicates which data path between the CPU and SSD will be cleared. For example, if an error occurs during a SSD write data transfer, the error is reported via the HSC status channel. A subsequent HSC command with the Read/write bit set and the Clear 100 Mbyte Channel Error bit set results in a Clear Channel signal sent over both the SSD input side and the CPU output side of the 100 Mbytes per second channel. The command also resets the HSC write logic. The clear operation does not result in a status word input. Any SSD read operation in progress at the time is not affected.

When the Clear 100 Mbyte Channel Error bit is set, all other command fields except the Read/write bit are ignored. In addition, the Clear Error command is ignored by the HSC if the error code is zero.

Diagnostic mode field

This 4-bit field is for maintenance purposes only. During normal operation, this field must contain all zeros.

Transfer length field

The 8-bit transfer length (XL) field designates the number of 64-word blocks of data to be transferred over the 100 Mbytes per second channel. From 1 to 256 blocks of data can be transferred with a single request. A field of all zeros in the command word indicates the maximum transfer length of 256 blocks (16,384 words). Figures 3-4 and 3-5 illustrate the mapping of address parameters into the three address words transmitted from the HSC to the CPU side of the 100 Mbytes per second channel and to the SSD port interface.

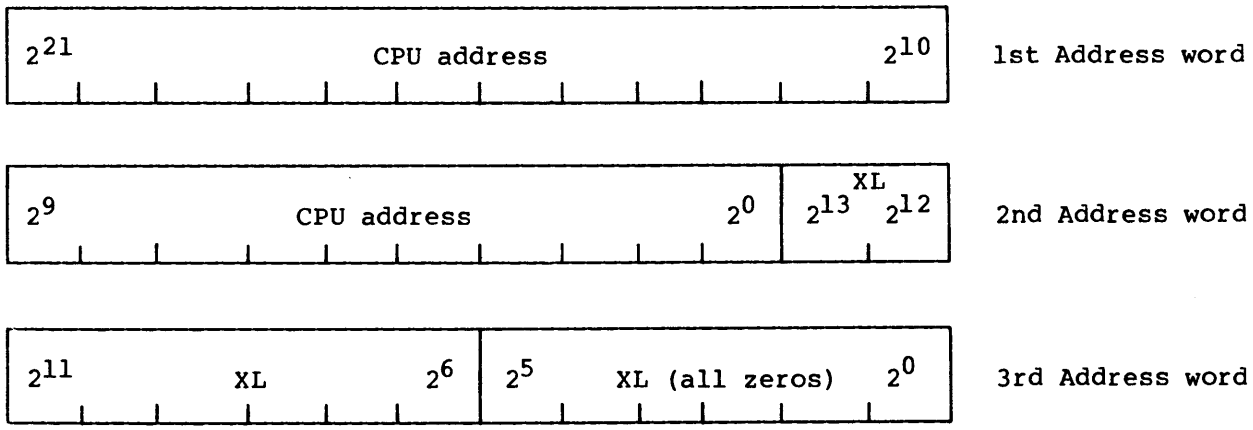


Figure 3-4. Address words sent from HSC to CPU side of 100 Mbytes per second channel

CPU address field

The 22-bit CPU address field contains the starting CPU address for the indicated transfer. Transfers to or from Central Memory can begin at any word address.

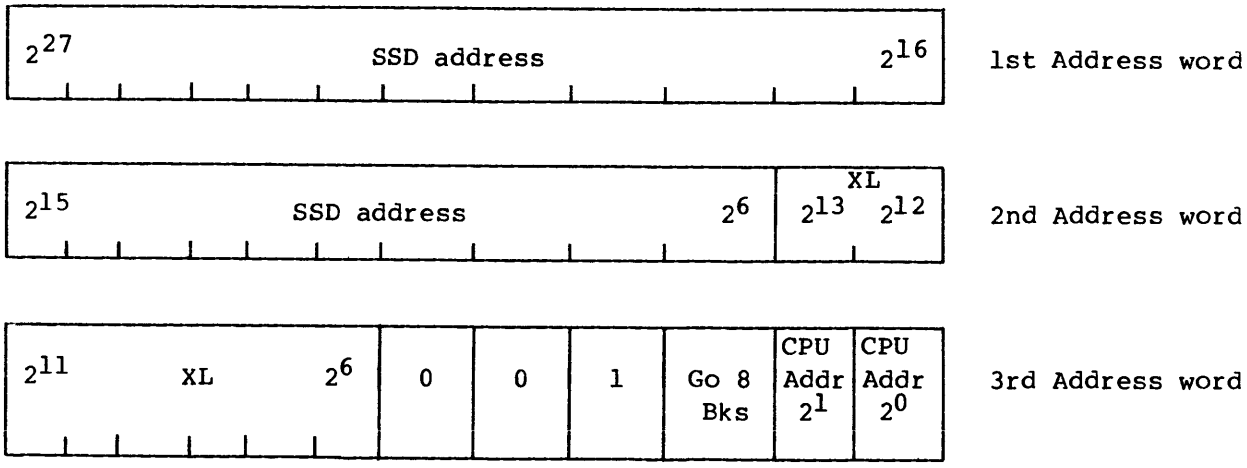


Figure 3-5. Address words sent from HSC to SSD port interface

SSD address field

The 22-bit SSD address field contains the starting SSD address divided by 100_8.

HSC STATUS CHANNEL

Upon completing an SSD data transfer or upon detecting an error condition, a single 64-bit status word (figure 3-6) is transmitted from the HSC to a special mainframe CPU input channel designated the HSC status channel. An interrupt request is generated upon receiving the status word (see subsection on HSC status word). The HSC status channel occupies one CPU input channel and differs from a normal CRAY-1 S or CRAY-1 M CPU input channel in the following respects.

Channel limit address: Since all transfers over the HSC status channel consist of a single CPU word, the Channel Limit (CL) register is not implemented.

Programmable master clear: The programmable master clear sequence used for the normal CPU I/O channels is not used for the HSC status channel. Instead, executing a single Set CL instruction (0011*jk*) results in transmitting a 100-nanosecond master clear pulse to the HSC, causing the HSC to reset its circuitry

Programmable master clear: (continued)

and to pass the master clear pulse along to both the CPU and SSD sides of the 100 Mbytes per second channel. The HSC status channel is also reset in the process. The command channel is not affected by this operation.

HSC STATUS WORD

The format of the status word received from the HSC is shown in figure 3-6, and the fields are described below.

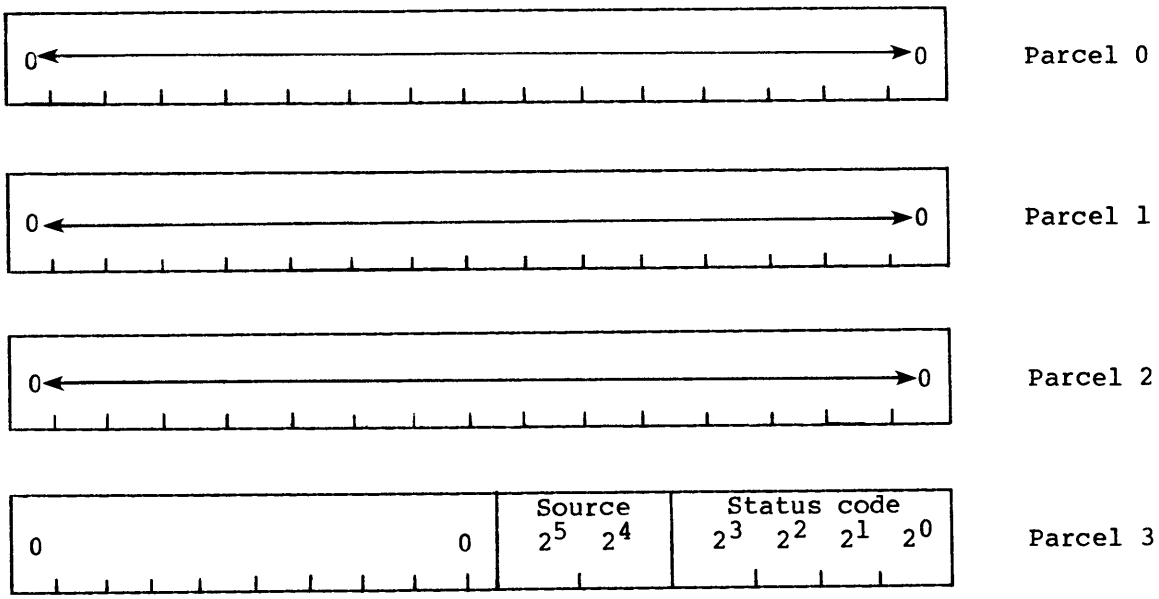


Figure 3-6. HSC status word format

Source field

The 2-bit source field indicates the operation pertaining to the accompanying status code. The source field is coded as follows:

Code	Function
00	Indicates an error was detected on the HSC command channel
01	Pertains to the most recently executed SSD read operation
10	Pertains to the most recently executed SSD write operation
11	Undefined

Status code field

The interpretation of the 4-bit status code field depends upon the source of the status word as indicated in the source field description above. Tables 3-1 through 3-3 list the possible status codes and their descriptions for command channel errors, read status, and write status, respectively.

Table 3-1. Command channel error status; source field = 00_2

Status code (binary)	Name	Description
00x1	Command parity error	A parity error was detected by the HSC in the HSC command word; command rejected.
001x	Command channel error	An error in the HSC command channel protocol was detected by the HSC; command rejected.

Table 3-2. SSD read status codes; source field = 01_2

Status code (octal)	Name	Description
00	Transfer complete	SSD read data transfer completed normally
01	Function error	A command to initiate a new SSD read transfer or a command to clear a read error was received before the previous read operation completed or before it was terminated by an error.
02	SSD active error	The SSD output side of the 100 Mbytes per second channel went inactive before sending Last Word flag while the CPU side was still active, or the SSD port enable switch is in the disable position.

Table 3-2. SSD read status codes; source field = 01_2 (continued)

Status code (octal)	Name	Description
03	SSD Transmit Address timeout	An SSD read transfer was attempted but the SSD output side of the 100 Mbytes per second channel failed to respond with a Transmit Address signal within three milliseconds, or the SSD channel enable switch is in the disable position.
04	SSD address error	The SSD output side of the 100 Mbytes per second channel received greater than or less than three Address Ready pulses, or there was a parity error in one of the address words transferred to the SSD.
05	SSD data error	The SSD detected a multiple-bit error upon reading memory.
06		Not used
07	SSD Data Ready timeout	The SSD output side of the 100 Mbytes per second channel is active but has not transmitted any data for three milliseconds.
10		Not used
11		Not used
12	CPU active error	The CPU input side of the 100 Mbytes per second channel went inactive while the SSD side of the channel was still active.
13	CPU Transmit Address timeout	An SSD read transfer was attempted but the CPU input side of the 100 Mbytes per second channel failed to respond with a Transmit Address within three milliseconds.
14	CPU address error	The CPU input side of the 100 Mbytes per second channel received greater than or less than three Address Ready pulses, or there was a parity error in one of the address word transfers to the CPU.

Table 3-2. SSD read status codes; source field = 01_2 (continued)

Status code (octal)	Name	Description
15	CPU data/block length error	The CPU input side of the 100 Mbytes per second channel detected a multiple-bit error in the data received from the SSD, or the last word of data was received and the block length count was not zero, or the last word of data was not received and the block length count was zero, or the CPU side received data when its data buffers were full.
16	CPU Last Word timeout	The last word of data was sent from the SSD output side of the 100 Mbytes per second channel but the CPU side failed to go inactive within three milliseconds.
17	CPU Transmit Data timeout	The Transmit Data signal from the CPU input side of the 100 Mbytes per second channel was not received by the HSC for three milliseconds while active.

Table 3-3. SSD write status codes; source field = 10_2

Status code (octal)	Name	Description
00	Transfer complete	SSD write data transfer completed normally.
01	Function error	A command to initiate a new SSD write transfer or a command to clear a write error was received before the previous write operation completed or before it was terminated by an error.
02	SSD active error	The SSD input side of the 100 Mbytes per second channel went inactive while the CPU side was still active, or the SSD port enable switch is in the disable position.

Table 3-3. SSD write status codes; source field = 10_2 (continued)

Status code (octal)	Name	Description
03	SSD Transmit Address timeout	An SSD write transfer was attempted but the SSD input side of the 100 Mbytes per second channel failed to respond with a Transmit Address signal within three milliseconds, or the SSD channel enable switch is in the disable position.
04	SSD address error	The SSD input side of the 100 Mbyte channel received greater than or less than three Address Ready pulses, or there was a parity error in one of the address words transferred to the SSD.
05	SSD data/block length error	The SSD detected a multiple-bit error while writing data received by the SSD input side of the 100 Mbytes per second channel, or the last word of data was received by the SSD side and the block length count was not zero, or the last word of data was not received and the block length count was zero, or the SSD side received data when its data buffers were full.
06	SSD Last Word timeout	The last word of data was sent from the CPU output side of the 100 Mbytes per second channel but the SSD side failed to go inactive within three milliseconds.
07	SSD Transmit Data timeout	The Transmit Data signal from the SSD input side of the 100 Mbytes per second channel was not received by the HSC for three milliseconds while active.
10		Not used
11		Not used
12	CPU active error	The CPU output side of the 100 Mbyte channel went inactive before sending Last Word flag while the SSD side was still active.

Table 3-3. SSD write status codes; source field = 10_2 (continued)

Status code (octal)	Name	Description
13	CPU Transmit Address timeout	An SSD write transfer was attempted but the CPU output side of the 100 Mbytes per second channel failed to respond with a Transmit Address within three milliseconds.
14	CPU address error	The CPU output side of the 100 Mbyte channel received greater than or less than three Address Ready pulses, or there was a parity error in one of the address word transfers to the CPU.
15	CPU data error	The CPU detected a multiple-bit error upon reading memory.
16		Not used
17	CPU Data Ready timeout	The CPU output side of the 100 Mbyte channel is active but has not transmitted any data for three milliseconds.

SSD PROGRAMMING

The following programming notes apply when configuring an SSD with a CRAY-1 S or a CRAY-1 M Series Computer System.

CHANNEL INSTRUCTIONS

Table 3-4 lists the I/O instructions implemented on the CPU. Each instruction's use is summarized for normal I/O channel pairs and for the HSC command/status channel pair.

Table 3-4. Channel instruction summary

I/O instruction	Normal usage	Command/status channel usage
†0010*jk*	Set CA of channel (A*j*) to (A*k*); activate channel; interrupt when transfer is complete.	Set CA of channel (A*j*) to (A*k*), set CL to (A*k*)+1, activate channel. HSC status channel interrupts when status word transfer is done. HSC command channel never interrupts.
†0011*jk*	Set CL of channel (A*j*) to (A*k*).	Master clear HSC via HSC status channel (A*j*). Ignored by command channel.
†0012*jx*	Clear Interrupt flag and Error flag on channel (A*j*) and deactivate channel.	Clear Interrupt flag and Error flag on HSC status channel (A*j*) and deactivate channel. Clear HSC Command Channel Busy flag on HSC command channel (A*j*) and deactivate channel.
033*i*0*x*	Channel number of highest priority interrupt request to A*i*.	Channel number of highest priority interrupt request to A*i*.
033*ij*0	CA of channel (A*j*) to A*i*.	CA of channel (A*j*) to A*i*.
033*ij*1	Error flag of channel (A*j*) to A*i*.	Error flag of HSC status channel (A*j*) to A*i*. Channel Busy flag of HSC command channel (A*j*) to A*i*.

† Privileged to Monitor Mode

CHANNEL MASTER CLEAR SEQUENCE

The HSC command/status channel pair can be master cleared with the following instruction sequence.

Octal code	CAL syntax		Description
022*ijk*	A1	ICH	HSC status channel number
022*ijk*	A2	OCH	HSC command channel number
0011*jk*	CL,A1	A*k*	Master Clear HSC status channel, HSC, and 100 Mbytes per second channel
0012*jk*	CI,A2		Deactivate HSC command channel

After executing this sequence, neither the HSC command channel nor the HSC status channel should be activated for 8 CPs. This sequence clears the HSC command channel, the HSC status channel, the HSC, and the SSD and CPU control channels of the 100 Mbytes per second channel connected to the HSC.

This operation differs from transmitting a Clear 100 Mbyte Channel Error command. In this operation, the master clear sequence is all encompassing, whereas the Clear 100 Mbyte Channel Error command only clears the selected 100 Mbytes per second channel data path, that is, one input channel, one output channel, and a portion of the HSC logic.

HSC PROGRAMMING SEQUENCE

The HSC provides only one I/O interrupt request (generated by the HSC status channel) for each data transfer. Two program sequences are involved in communicating with the HSC. First, a command must be sent to the HSC. Subsequently, the HSC status channel interrupt request must be processed upon completing the data transfer.

HSC command sequence example

An HSC command sequence example is listed below. Upon activating the command channel, the channel remains active for approximately 48 CPs, assuming no Central Memory conflicts. If the HSC command channel is active for a substantially longer period of time, the channel can be assumed to be hung. For this reason, the timeout check in the sample sequence checks the channel's active time.

Octal code	CAL syntax		Description
022206	CMDSEQ A2	6	Timeout value; approximately 1 usec.
0221*jk*	A1	OCH	HSC command channel number
033011	CHKCMD A0	CE,A1	Sample Command Channel Busy flag
011*ijkm*	JAZ	OUTCMD	Jump if channel not busy
031020	A0	A2-1	Else, wait for not busy
031220	A2	A2-1	
011*ijkm*	JAN	CHKCMD	Loop if timeout not expired
006*ijkm*	J	ERR	Exit; timeout error.
0202*jkm*	OUTCMD A2	CMD	Address of command word
001012	CA,A1	A2	Set CA and activate HSC command channel
006*ijkm*	J	EXIT	

HSC status channel interrupt routine example

An example of an HSC status channel interrupt routine is listed below. This example assumes the CPU responded to the HSC status channel interrupt request with an exchange jump into the interrupt handler. If it is preferrable to remain in Monitor Mode after executing the command sequence, the HSC Status Channel address register can be monitored to indicate completion of the transfer. An SSD transfer of maximum block length (256 blocks or 16,384 words) completes in approximately 1.25 milliseconds.

Octal code	CAL syntax		Description
033200	A2	CI	Read channel number of interrupt request
030002	A0	A2	
010*ijkm*	JAZ	EXIT	Exit if not I/O interrupt
0221*jk*	A1	ICH	HSC status channel number
031021	A0	A2-A1	
011*ijkm*	JAN	EXIT	Exit if interrupt not from HSC
1207*jkm*	S7	STAT,0	Get status word
033011	A0	CE,A1	Read Status Channel Error flag
011*ijkm*	JAN	ERR	Exit; channel error.
001210	CI,A1		Clear status channel interrupt request
0202*jkm*	A2	STAT	Address of status buffer
001012	CA,A1	A2	Reactivate HSC status channel
Process status word.			

INDEX

INDEX

100 Mbytes per second channel, 1-4, 3-1
    programming, 3-3
1250 Mbytes per second channel, 1-4, 2-1
    programming, 2-3

Address words, 3-6

Block length registers, 2-4
Buffers, 2-2

Central Memory, 2-2
Central Memory address, 2-3
Channel Error flag, 3-4
Channel instruction summary, 3-14
Channel instructions, 3-13
Channel limit address, 3-4, 3-7
Channel master clear sequence, 3-15
Channels
    100 Mbytes per second channel, 1-4, 3-6
    1250 Mbytes per second channel, 1-4, 2-1
    I/O, 2-1
Clear 100 Mbytes per second channel error
  field, 3-5
Clear interface/abort sequence, 2-4
Command channel, 3-1, 3-3
    command sequence example, 3-15
    differences from CPU output channel, 3-3
    error status, 3-9
    HSC command word, 3-4
Command Channel Busy flag, 3-4
Command Word, 3-4
    clear 100 Mbytes per second channel
      error field, 3-5
    CPU address field, 3-6
    CPU loopback field, 3-5
    diagnostic mode field, 3-6
    format, 3-5
    read/write field, 3-4
    SSD address field, 3-7
    transfer length field, 3-6
Configurations, 2-1, 3-2
Control paths between SSD and CRAY-1 M
  mainframe, 3-3
Control paths between SSD and CRAY-1 S
  mainframe, 3-3
Control timing, 1-7
Conventions, 1-1
CPU address field, 3-6
CPU loopback field, 3-5
CRAY X-MP port, 1-4
CRAY X-MP Series Computer System with SSD,
  2-1
CRAY-1 M port, 1-4
CRAY-1 S port, 1-4
CRAY-1 S Series Computer System with SSD,
  3-2

Data paths between SSD and CRAY-1 M
  mainframe, 3-3
Data paths between SSD and CRAY-1 S
  mainframe, 3-3
Data protection, 1-3, 1-5
Data transfer, 2-2
    64-word block, 1-6, 2-3
    burst rate, 1-3, 2-1
Diagnostic mode field, 3-6

Error correction/error detection, 1-5
Error Log, 1-3, 1-5

High speed controller (HSC), 3-1
    command sequence example, 3-15
    programming sequence, 3-15
    status channel interrupt routine
      example, 3-16
HSC, see High speed controller

I/O channels, 2-1
I/O port, 2-2
Instructions, 2-3, 3-14
Internal memory refresh port, 1-4
Interrupt request, 3-3

Maintenance port, 1-4
Master clear after power on, 2-5
Memory Channel, 1-4

Port interface, 3-7
Ports, 1-3
    Port 0 - Internal Memory Refresh, 1-4
    Port 1 - Maintenance Port, 1-4
    Port 2 - CRAY-1 S and CRAY-1 M Port, 1-4
    Port 3 - CRAY X-MP Port, 1-4
Programmable master clear, 3-7

Read/write field, 3-4
Read operation, 2-2
Read status codes, 3-9

SECDED, see Error correction/error detection
Source field, 3-8
SSD
    chassis, 1-2
    description, 1-1
    memory, 1-5, 2-2
       addressing, 1-6
       control, 1-6
       cycle, 1-7
       organization, 1-6
       ports, 1-3, 2-2
       size, 1-3, 1-5, 1-7
    operating characteristics, 1-3
    programming, 2-3, 3-13
       block length registers, 2-4
       Central Memory address, 2-3
       channel instructions, 3-13
       channel master clear sequence, 3-15
       clear interface/abort sequence, 2-4
       HSC command sequence example, 3-15
       HSC programming sequence, 3-15
       HSC status channel interrupt routine
         example, 3-16
       master clear after power on, 2-5
       SSD block address, 2-3
       status word error flags, 2-4
       transfer block length read/write
         mode, 2-3
    read operation, 2-2
    read status codes, 3-9
    write operation, 2-2
    write status codes, 3-11
SSD address field, 3-7
SSD and CRAY X-MP operation, 2-1
SSD and CRAY-1 M operation, 3-1
SSD and CRAY-1 S operation, 3-1
SSD block address, 2-3
SSD read status codes, 3-9
SSD write status codes, 3-11
Status channel, 1-5, 3-1, 3-7
    differences from CPU input channel, 3-7
    HSC status word, 3-8
    interrupt routine example, 3-16
Status code field, 3-9
Status word, 3-8
    format, 3-8
    source field, 3-8
    status code field, 3-9
Status word error flags, 2-4

Transfer block length read/write mode, 2-3
Transfer length field, 3-6

Write operation, 2-2
Write status codes, 3-11

READERS COMMENT FORM

Solid-state Storage Device (SSD) Reference Manual HR-0031

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.

CUT ALONG THIS LINE

FOLD

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.

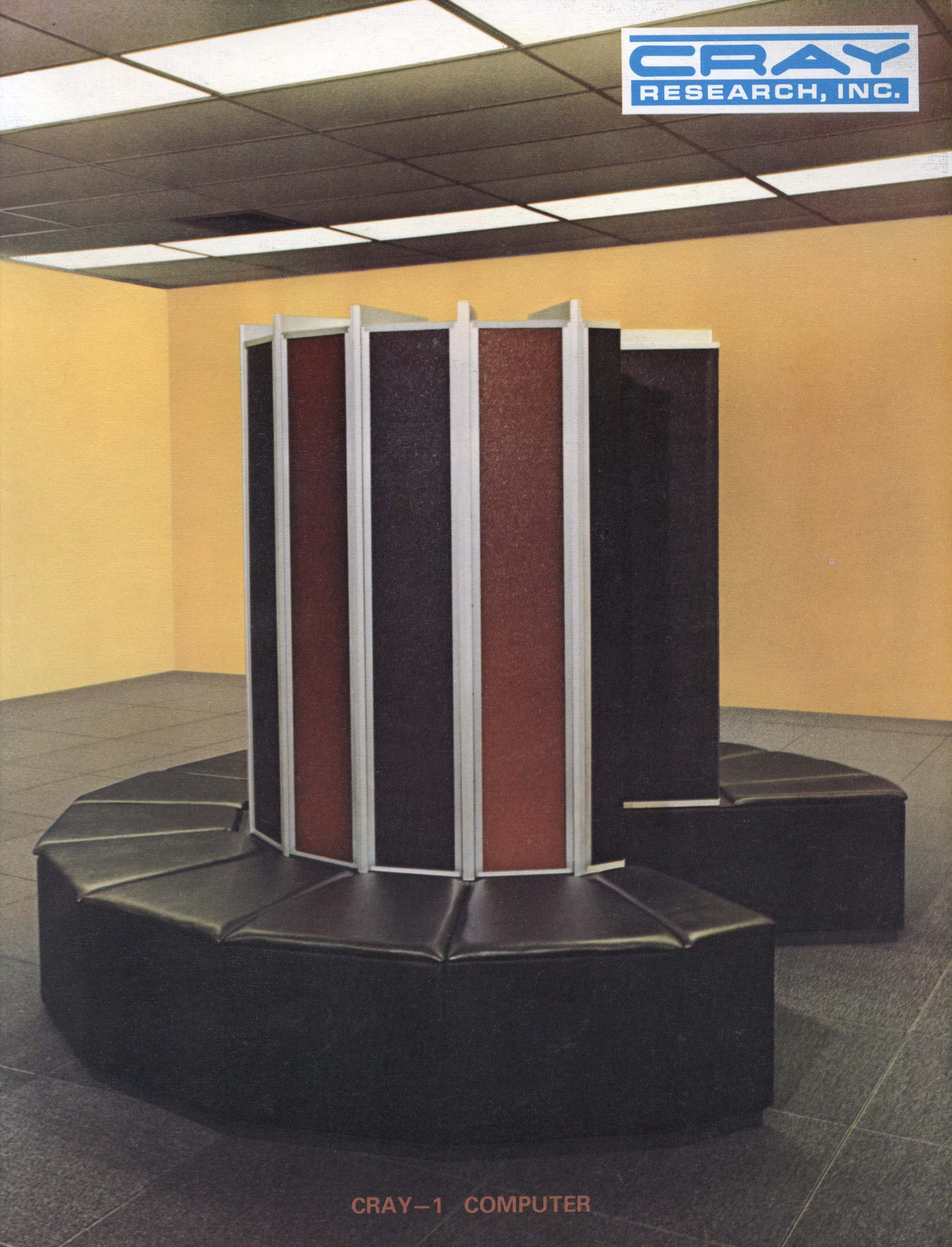
Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

FOLD

STAPLE

Cray Research, Inc.
Publications Department
1440 Northland Drive
Mendota Heights, MN 55120
612-452-6650
TLX 298444



CRAY–1 COMPUTER

AN INTRODUCTION

TO THE

CRAY-1 COMPUTER

Copyright © 1975 by Cray Research, Inc.

The Cray Research CRAY-1 computer is a powerful general purpose computer incorporating scalar and vector capabilities and a large, fast bi-polar memory. Vector processing provides result rates greatly exceeding the result rates of conventional scalar processing. Several unique design features of the CRAY-1 provide formidable competition for other existing vector machines. This paper describes the structure of the CRAY-1 and enumerates some of the important unique design features which give the machine its power. The paper concludes with examples to illustrate the speed of the CRAY-1 in vector applications.

Operating registers

The primary operating registers of the CRAY-1 are the scalar and vector registers, called S and V registers, respectively. Each of the eight V registers has sixty-four elements. A scalar instruction may perform some function, such as addition, obtaining operands from two S registers and entering the result into another S register. The analagous vector instruction performs the same function, obtaining each clock period (12.5 nanoseconds) a new pair of operands from two V registers. Results are entered into elements of another V register. The contents of the vector length (VL) register determines the number of operations performed by the vector instruction. Eight 24-bit A registers are used as address registers for memory references and as index registers. The A and S registers are each supported by sixty-four rapid-access temporary storage registers, called B and T registers, respectively. Data can be transferred between the A, B, S, T, or V registers and memory.

Memory

The CRAY-1 memory is constructed of bi-polar 1024-bit LSI chips. Up to one million 64-bit words are arranged in 16 banks with a bank cycle time of four clock periods. The short cycle time provides an extremely efficient random-access memory. One parity bit per word is maintained in 16 modules of the CPU. There is no inherent memory degradation for machines with less than one million words of memory.

Instruction buffers

All instructions, which may be 16 or 32 bits, are executed from four instruction buffers, each consisting of sixty-four 16-bit registers. Associated with each instruction buffer is a base address register that is used to determine if the current instruction resides in a buffer. Since the four instruction buffers are large, substantial program segments may reside in the buffers. Forward and backward branching within the buffers is possible and the program segments may be discontiguous. When the current instruction does not reside in a buffer, one of the instruction buffers is filled from memory. Four memory words are read per clock period to the least recently filled instruction buffer. To allow the current instruction to issue as soon as possible, the memory word containing the current instruction is among the first to be read.

Input/Output

There are twenty-four I/O channels, twelve of which are input and twelve output. Any number of channels may be active at a given time. Each channel has a maximum transfer rate of 640 megabits. At most one 64-bit word per clock period can be transferred to or from memory and this is attained when four input channels and four output channels are simultaneously operating at their maximum rate. In practice this theoretical transfer rate is limited by the speed of peripheral devices and by memory reference activity of the CPU.

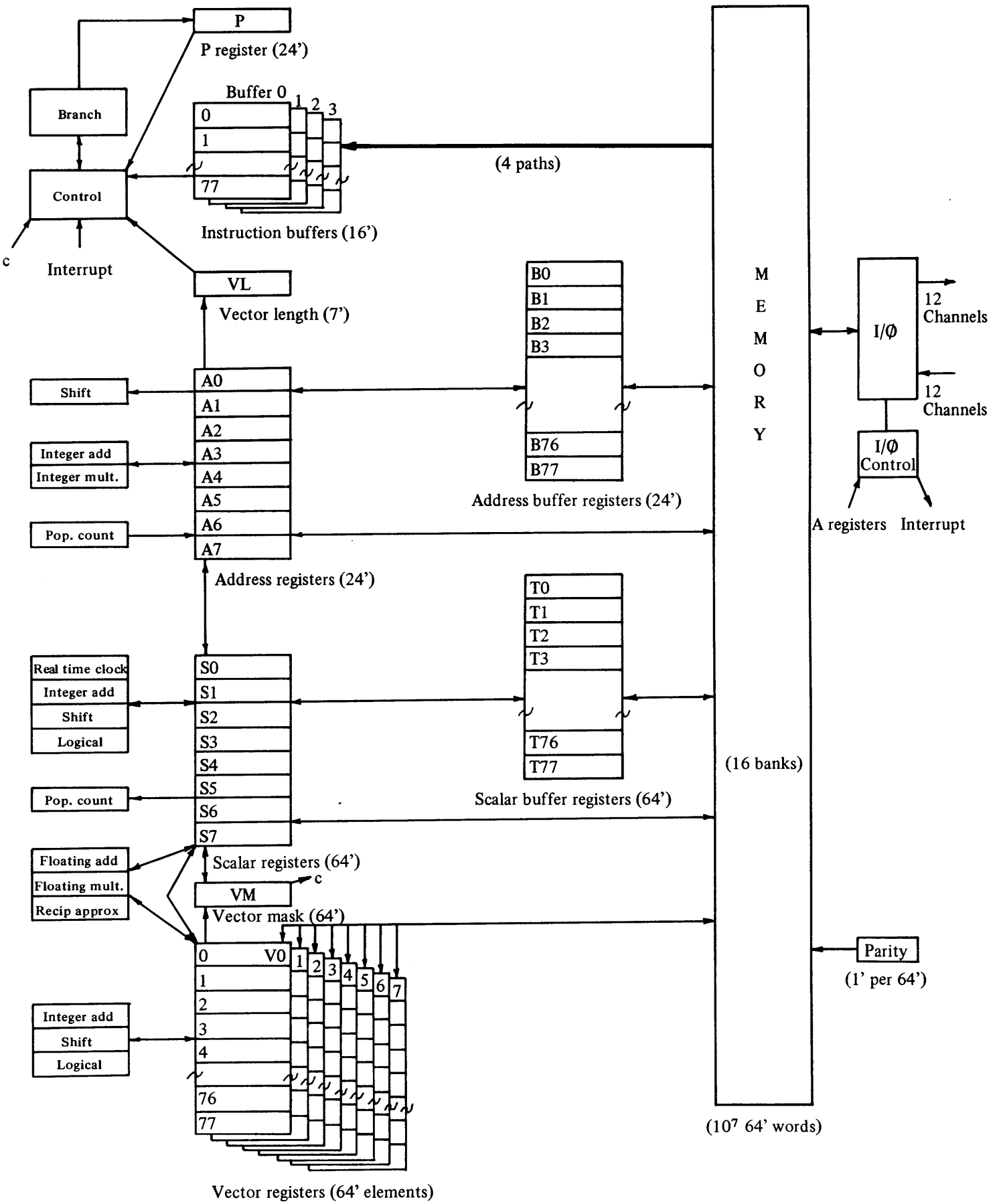


Figure 1. Registers block diagram

Functional units

There are twelve functional units in the CPU. Each is a specialized unit implementing algorithms for a portion of the instructions. Each unit is independent of the other units and a number of functional units may 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. There is no information retained in a functional unit for reference in subsequent instructions. These units operate essentially in three-address mode with limited source and destination addressing.

Three functional units provide 24-bit results to the A registers only:

— integer add
— integer multiply
— population count

Three functional units provide 64-bit results to the S registers only:

— integer add
— shift
— logical

Three functional units provide 64-bit results to the V registers only:

— integer add
— shift
— logical

Three functional units provide 64-bit results to either the S or V registers:

— floating add
— floating multiply
— reciprocal approximation

Integer arithmetic is performed in 2's complement mode. Floating point quantities have signed magnitude representation.

All functional units are fully segmented. This means that the information arriving at the unit, or moving within the unit, is captured and held in a new set of registers at the end of every clock period. It is therefore possible to start a new set of operands for unrelated computation into a functional unit each clock period even though the unit may require more than one clock period to complete the calculation. 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. Functional units servicing the vector instructions produce one result per clock period.

Vector instructions

Vector instructions may be classified into four types. One type of vector instruction obtains operands from one or two V registers and enters results into another V register (Fig. 2a). Successive operand pairs are transmitted from Vj and Vk to the segmented functional unit each clock period and the corresponding results emerge from the function unit n clock periods later. n is constant for a given functional unit and is called the functional unit time. Results are entered into result register Vi. The contents of the vector length (VL) register determines the number of operand pairs processed by the functional unit.

The second type of vector instruction obtains one operand from an S register and one from a V register (Fig. 2b). A copy of the S register is transmitted to the functional unit with each V-register operand.

The last two types of vector instruction transmit data between memory and the V registers (Fig. 2c and 2d). A path between memory and the V registers may be considered a functional unit for timing considerations.

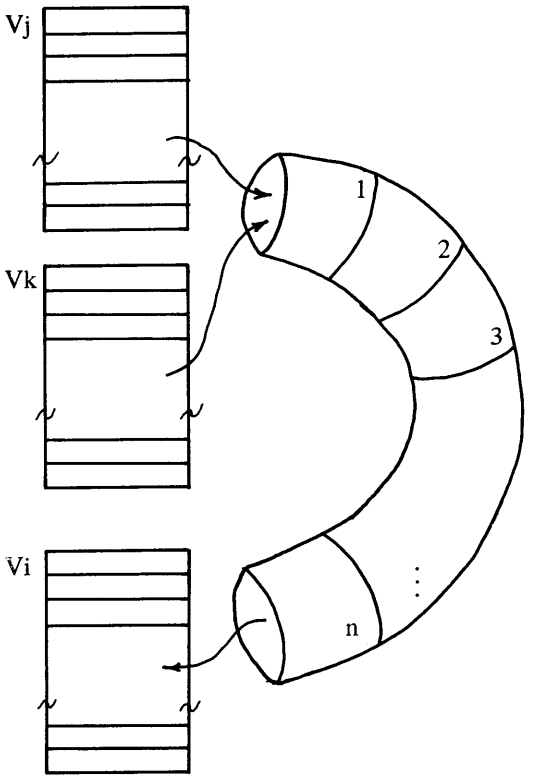


Figure 2a. Type I vector instruction

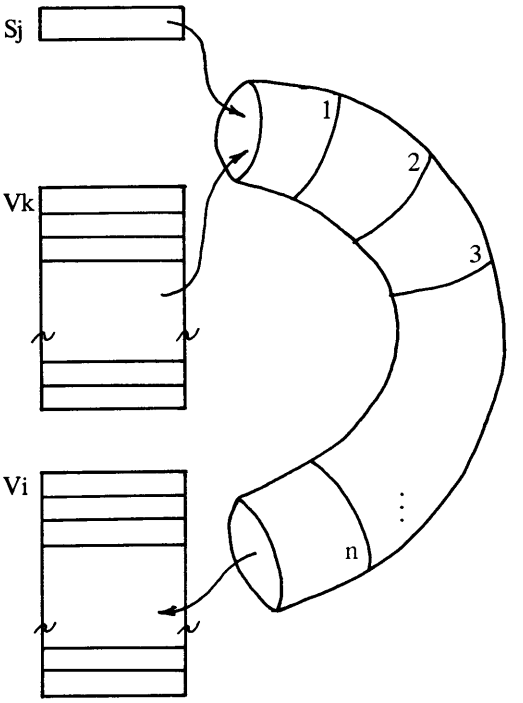


Figure 2b. Type II vector instruction

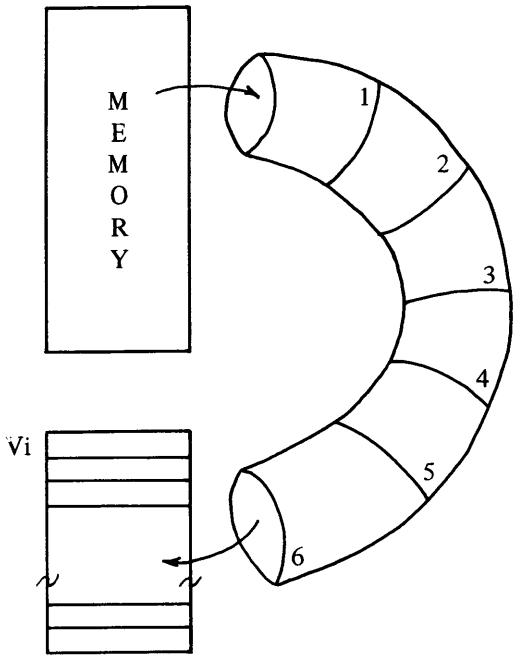


Figure 2c. Type III vector instruction

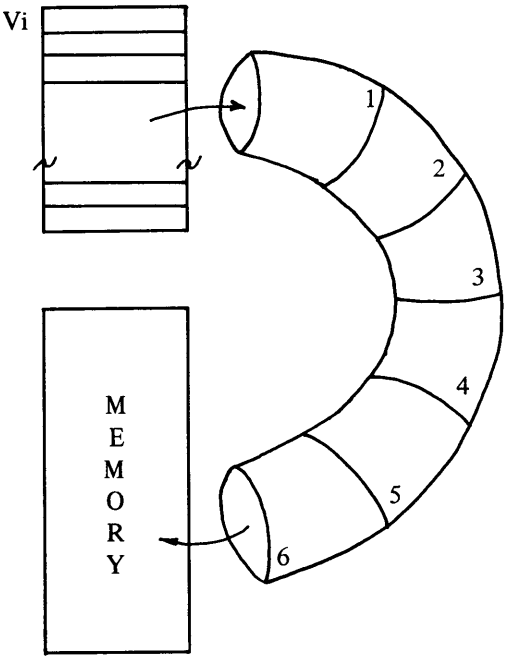


Figure 2d. Type IV vector instruction

It is important to understand functional unit segmentation, especially as it relates to execution of vector instructions. Let a particular element of a V register be specified by adding the element number as a subscript to the register name. For example, the first three elements of register V1 are $V1_0$, $V1_1$, and $V1_2$, respectively. Since a vector register has sixty-four elements, the last element of V1 is $V1_{63}$. Figure 3 shows a timing chart for execution of a floating point addition instruction. This instruction is type I since operands are obtained from two vector registers. When the instruction issues at clock period t_0, the first pair of elements ($V1_0$ and $V2_0$) is transmitted to the add functional unit where it arrives at clock period t_1. The dashed lines indicate transit to and from the functional unit. The functional unit time for this unit is six clock periods, so the first result, which is the sum of $V1_0$ and $V2_0$, exits from the functional unit at clock period t_7. The sum is transmitted to the first element of result register V0 arriving at clock period t_8. Because the functional unit is fully segmented, the second pair of elements ($V1_1$ and $V2_1$) is transmitted to the add functional unit at clock period t_1. At clock period t_2 the functional unit is in the process of performing two additions simultaneously, since the addition of $V1_0$ and $V2_0$ was begun in the previous clock period. The second result, which is the sum of $V1_1$ and $V2_1$, is entered into the second element of result register V0 at clock period t_9. Continuing in this manner, a new pair of elements enters the functional unit each clock period and the corresponding sum emerges from the unit six clock periods later and is transmitted to the result register. Since a new addition is begun each clock period, six additions may be in progress at one time. In general, the number of operations which can be performed simultaneously by a functional unit is equal to the functional unit time.

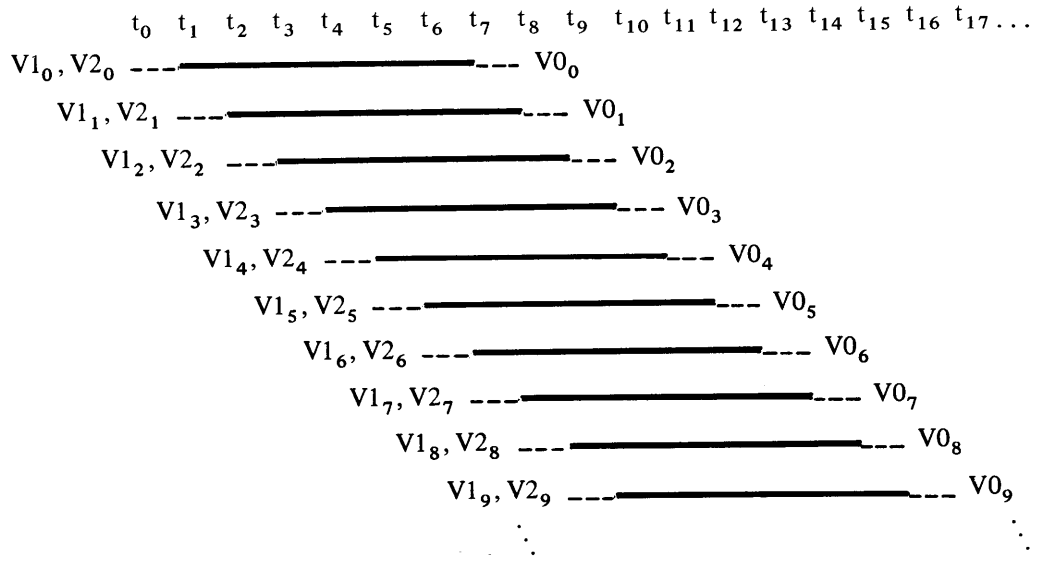


Figure 3. Vector instruction timing example (V0 ← V1 + V2)

The vector length determines the total number of operations performed by a functional unit. Although each vector register has sixty-four elements, only the number of elements specified by the vector length register is processed by a vector instruction. Vectors which have more than sixty-four elements are processed under program control, in groups of 64 (with a possible residue). A later section on vector loops will illustrate the processing of long vectors.

Functional unit and operand register reservations

When a vector instruction issues, the required functional unit and the operand registers are reserved for the number of clock periods determined by the vector length. A subsequent vector instruction which requires the same functional unit or operand register cannot issue until the reservations are released. When two vector instructions use different functional units and vector registers, they are independent and may issue in neighboring clock periods. Example 1a shows two independent instructions. Both execute concurrently with a one clock period difference in their issue times. Examples 1b-1d illustrate the effect of functional unit and operand register reservation when two instructions are not independent. Example 1b shows two add instructions. When the first instruction issues, the floating add functional unit and operand registers V1 and V2 are reserved. Issue of a second add instruction is delayed until the functional unit is free. Example 1c shows an add instruction followed by a multiply instruction. As in the previous example, the floating add functional unit and operand registers V1 and V2 are reserved when the first instruction issues. Issue of the second instruction is delayed until the operand register V1 is free. The second instruction in example 1d is delayed because of both functional unit and operand register reservations.

V0 ← V1 + V2
V3 ← V4 * V5

Ex. 1a. Independent instructions

V3 ← V1 + V2
V6 ← V4 + V5

Ex. 1b. Functional unit reservation

V3 ← V1 + V2
V6 ← V1 * V5

Ex. 1c. Operand register reservation

V0 ← V1 + V2
V3 ← V1 + V5

Ex. 1d. Functional unit and operand register reservation

Result register reservations and chaining

When a vector instruction issues, the result register is reserved for the number of clock periods determined by the vector length and functional unit time. This reservation allows the final operand pair to be processed by the functional unit and the corresponding result to be transmitted to the result register.

A result register becomes the operand register of a succeeding instruction. In the process called "chaining", the succeeding instruction issues as soon as the first result arrives for use as an operand. This clock period is termed "chain slot time" and it occurs only once for each vector instruction. If the succeeding instruction cannot issue at chain slot time because of a prior functional unit or operand register reservation, then it must wait until the result register reservation is released. Figure 4a shows a chain of four instructions which read a vector of integers from memory, add that vector to another, shifting the sum, and finally forming the logical product of the shifted sum and a mask vector. The result of the four instructions is in vector register V5. Figure 4b graphically depicts the passage of information through the functional units. The diagram illustrates that the functional units may be considered links in a chain which works as a whole to produce the final result.

V0 ← memory	(memory read)	V3 ← V2<A3	(left shift)
V2 ← V0+V1	(integer add)	V5 ← V3&V4	(logical product)

Figure 4a. Chaining example

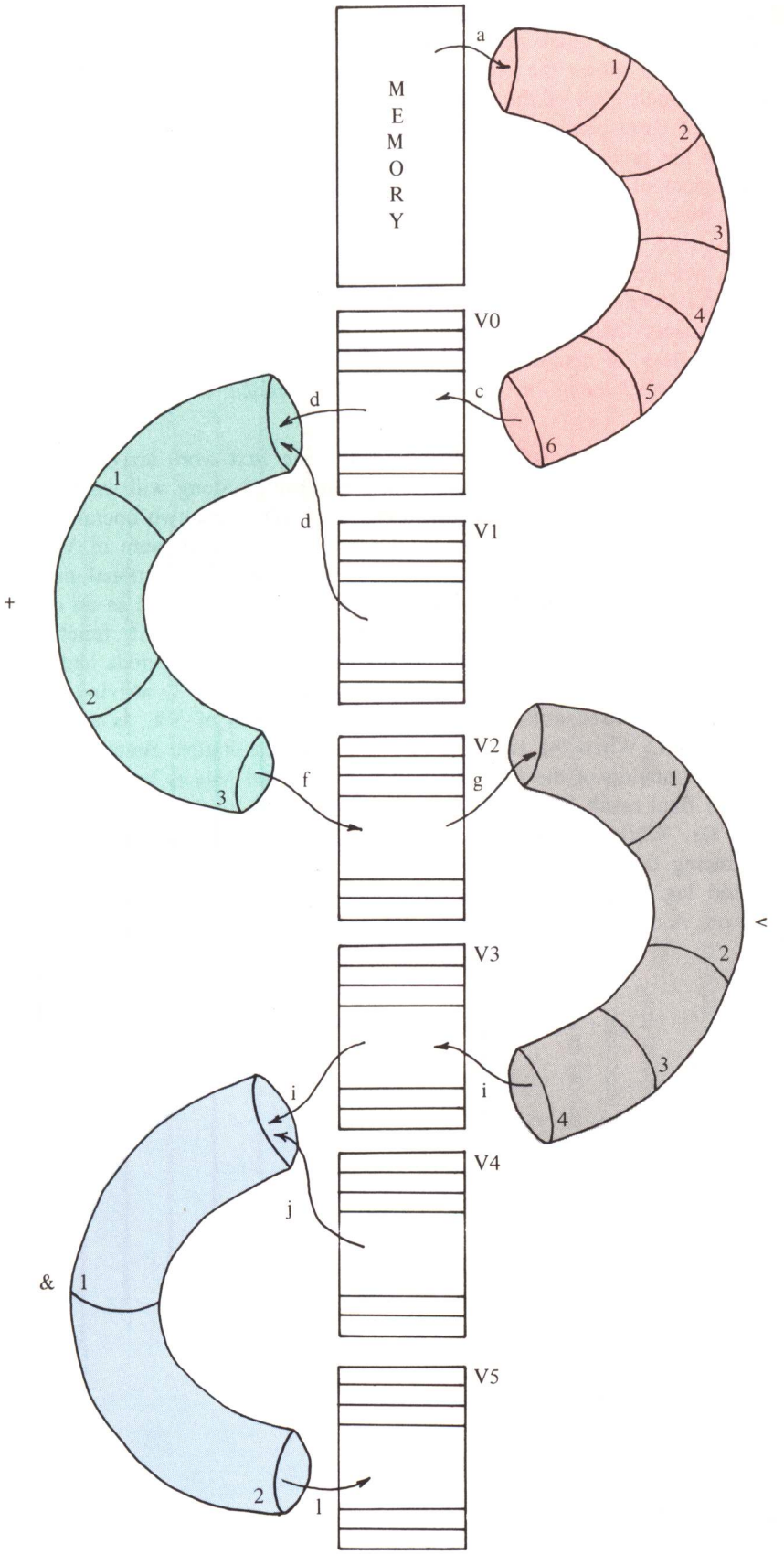
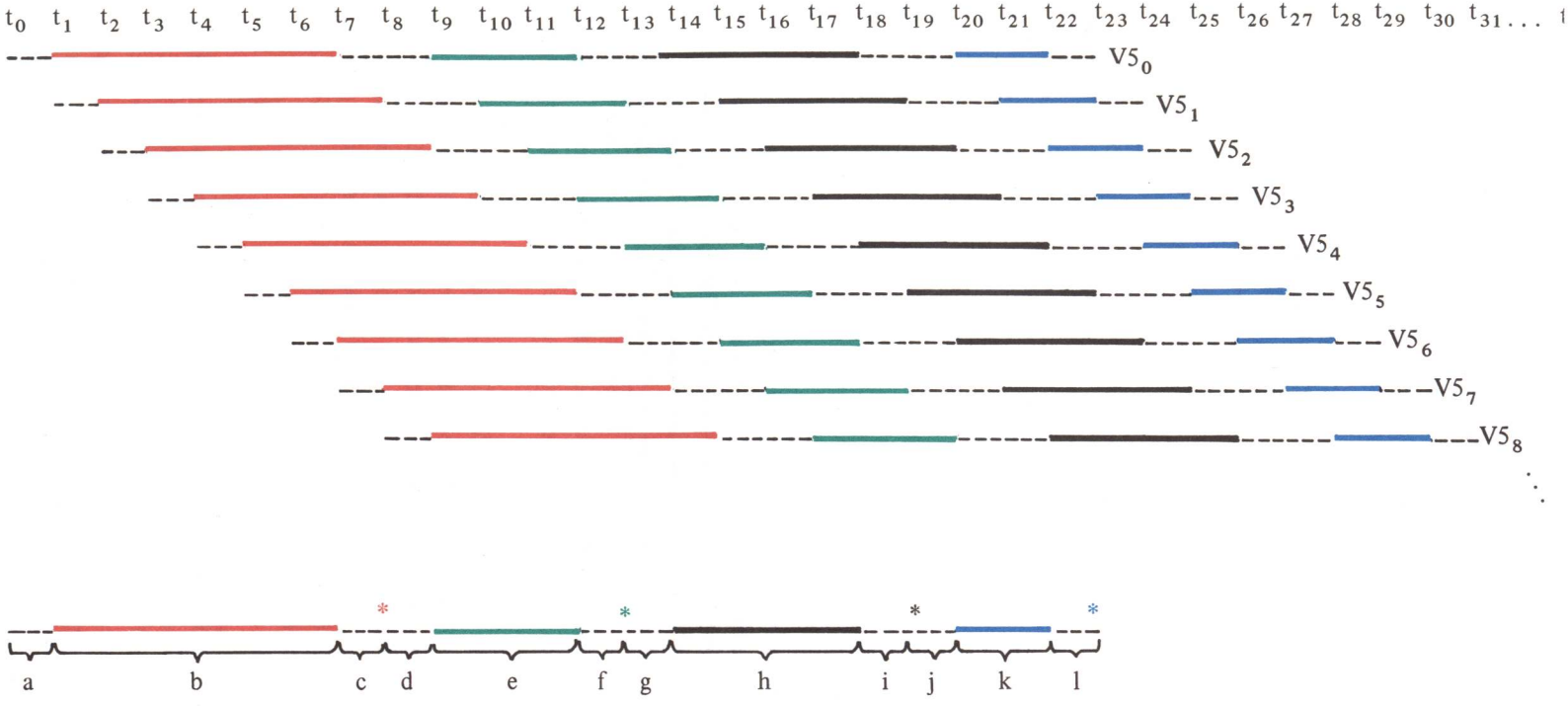


Figure 4b. Pictorial representation of chaining example

The timing diagram in Figure 4c will help clarify the concept of chaining. Gradations along the horizontal axis represent clock periods. The memory read instruction issues at clock period t_0. Each horizontal line shows the production of one element of the V5 result vector. Time spent in passing through each of the four functional units used in the instruction sequence is indicated by bars of corresponding length. The bars are color-coded to agree with Figures 4a and 4b. Note that the production of a new element of V5 begins each clock period. Production of the first element of V5 begins at clock period t_0 with the reading of the first word from memory, production of the second element of V5 begins at clock period t_1, with the reading of the second word from memory, and so on. The first result enters V5 at clock period t_{23} and a new result enters V5 each clock period thereafter. The first horizontal line, which shows the production of the first element of V5 ($V5_0$), is reproduced below the timing diagram with segments lettered for identification. Chain slot times for each functional unit are marked by asterisks. A detailed description of the production of $V5_0$ may be illustrative; production of the other elements of the result vector is identical except for the staggered start times.

The vector read instruction issues at clock period t_0. The first word arrives in element 0 of register V0 at clock period t_8, and is immediately transmitted along with element 0 of register V_1, as an operand to the integer add functional unit. When the two operands arrive at the integer add functional unit at clock period t_9, the computation of the sum of $V0_0$ and $V1_1$ is begun. Three clock periods later (t_{12}) the sum is sent from the functional unit to element zero of V2. It arrives at clock period t_{13} and is immediately transmitted as an operand to the shift functional unit. At clock period t_{14} the operand arrives at the shift functional unit and the shift operation is begun. The operation is completed four clock periods later (t_{18}) and the shifted sum is sent from the functional unit to element zero of V3, arriving the next clock period. It is immediately transmitted, along with element zero of V4, as an operand to the logical functional unit. When the two operands arrive at the logical functional unit at clock period t_{20}, the computation of the logical product of $V3_0$ and $V4_0$ is begun. Two clock periods later (t_{22}) the final result is sent from the functional unit to element zero of V5, arriving at clock period t_{23}. While all this has been going on, production of the second element of V5 has been tracing the same path through the vector registers and functional units with a one clock period lag. Production of the third element of V5 lags one more clock period behind, and so on. A new result arrives at the V5 result register each clock period.



a – transit of memory word to "read functional unit"

b – transit of memory word through "read functional unit"

c – transit of memory word from "read functional unit" to element of V0

d – transit of operand elements in V0 and V1 to integer add functional unit

e – computation of sum by integer add functional unit

f – transit of sum from integer add functional unit to element of V2

g – transit of operand element in V2 to shift functional unit

h – shift operation performed by shift functional unit

i – transit of shifted sum from shift functional unit to element of V3

j – transit of operand elements in V3 and V4 to logical functional unit

k – logical operation performed by logical functional unit

l – transit of final result to element of V5

Figure 4c. Timing diagram for chaining example

Vector loops

Long vectors are processed in segments since the vector registers of the CRAY-1 cannot accommodate vectors with more than 64 elements. The program construct created to process long vectors is called a vector loop. Each pass through the loop processes a 64-element (or smaller) segment of the long vectors. The general procedure is to compute the loop count based on the vector length before entering the loop. Inside the loop the program takes full advantage of the twelve independent functional units and chaining to read the current vector segments from memory, execute the required functions and return the results to memory. Loop control is performed in the scalar registers concurrently with vector processing. Loop branch time is hidden by the vector operations.

The following simple FORTRAN loop can be used to illustrate processing of long vectors:

```
      DØ    100   I = I, N
100   A(I) = 5.*B(I)+C
```

When N is 64 or less, all elements of the A array can be assigned a value with a sequence of seven instructions:

1.	S1 ← 5.	set constants	
2.	S2 ← C		
3.	VL← N	set vector length	
4.	V0 ← B array	read B array	chain
5.	V1 ← S1*V0	multiply elements by 5	chain
6.	V2 ← S2+V1	add C	chain
7.	A array ← V2	store A array	

Instructions 4-6 use different functional units (''memory'', multiply and add, respectively) and so they can be chained. When the V2 result register is free, the results are stored in the A array.

When N exceeds 64, a vector loop is required to generate the entire A array. Before entering the loop, N is divided by 64 to determine the loop count. If there is a remainder, less than 64 elements of A are generated in the first pass through the loop. The loop performs instructions 4-7 for a segment of the A and B arrays. The last vector operation at the bottom of the loop is to store the current segment of the A array in memory. This operation must be completed before the next segment of the B array can be read in the next pass through the vector loop. The time required to decrement the loop counter, increment the current position in the arrays and branch to the top of the loop is hidden because it is done in parallel with the store operation.

The concept of chaining and the use of register-to-register vector instructions are unique to the CRAY-1. The design minimizes the problem of speed degradation associated with memory-to-memory vector instructions. Start-up time for vector operations in the CRAY-1 are nominal; the benefits of vector processing are visible even for short vectors. The extensive concurrency attainable through the use of the 12 fully-segmented functional units can produce impressive results. A performance study of several subroutines for the CRAY-1 FORTRAN library and matrix multiplication illustrates this fact.

Fortran library

Each scalar FORTRAN library subroutine has a vector analog which employs the same algorithm in vector mode to produce several results at a time. The scalar subroutines must be called for each desired result, while the vector subroutines process an argument vector to obtain a vector of results. Performance studies on the CRAY-1 indicate that the vector subroutine outperforms its scalar counterpart whenever a vector of two or more results is required. Figure 5 depicts the behavior of the scalar and vector subroutines for several library functions. The cost of a result (in clock periods) is graphed as a function of vector length. The cost is constant for scalar subroutines since they must be called for each desired result; however, for vector subroutines the cost drops dramatically and rapidly approaches a lower limit as vector length increases. In all cases the vector cost is less than the scalar cost when more than one result is produced.

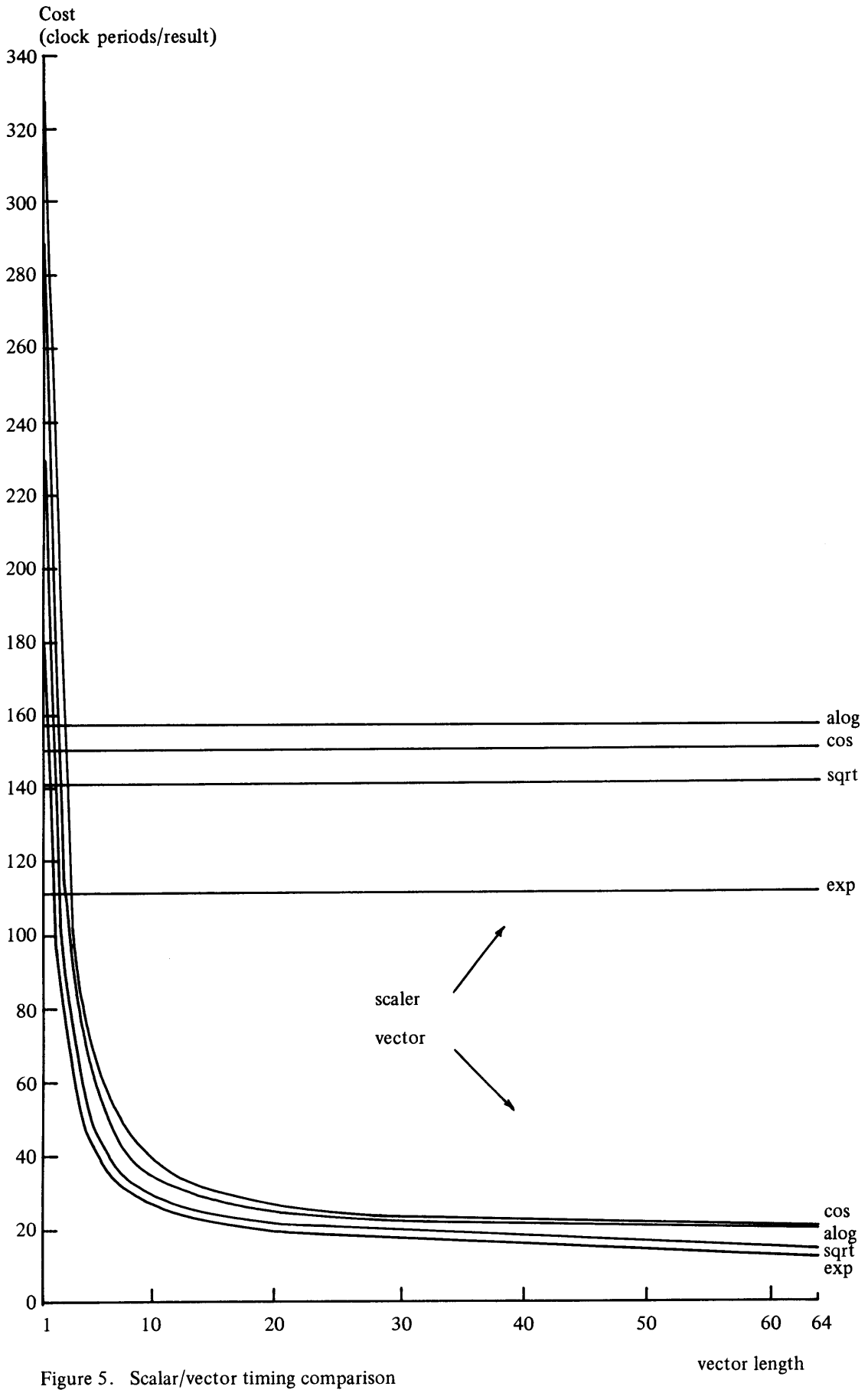


Figure 5. Scalar/vector timing comparison

Matrix multiplication

Let [X] denote a matrix and let the element in row i, column j be denoted by x_{ij}. Given matrix [A] of dimension K by N and the matrix [B] of dimension N by M, the product matrix [C] = [A] · [B] is defined by

$$c_{ij} = \sum_{n=1}^{N} a_{in} \cdot b_{nj}$$

Calculation of the product matrix is amenable to vector processing. The combination of multiplication and addition lends itself well to chaining. Figure 6 shows the CRAY-1 execution rate for multiplication of square matrices as a function of matrix dimension. The execution rate is defined in terms of "millions of floating point operations per second" (MFLOPs). This measure is more meaningful than the classical "millions of instructions per second" (MIPs), especially when comparing relative speeds of scalar and vector machines; a single vector instruction is equivalent to a loop of several scalar instructions. The number of floating point operations required to multiply two n-dimensional square matrices is $n^2 \cdot (2n-1)$, since each of the n^2 elements of the result matrix is formed by summing n products.

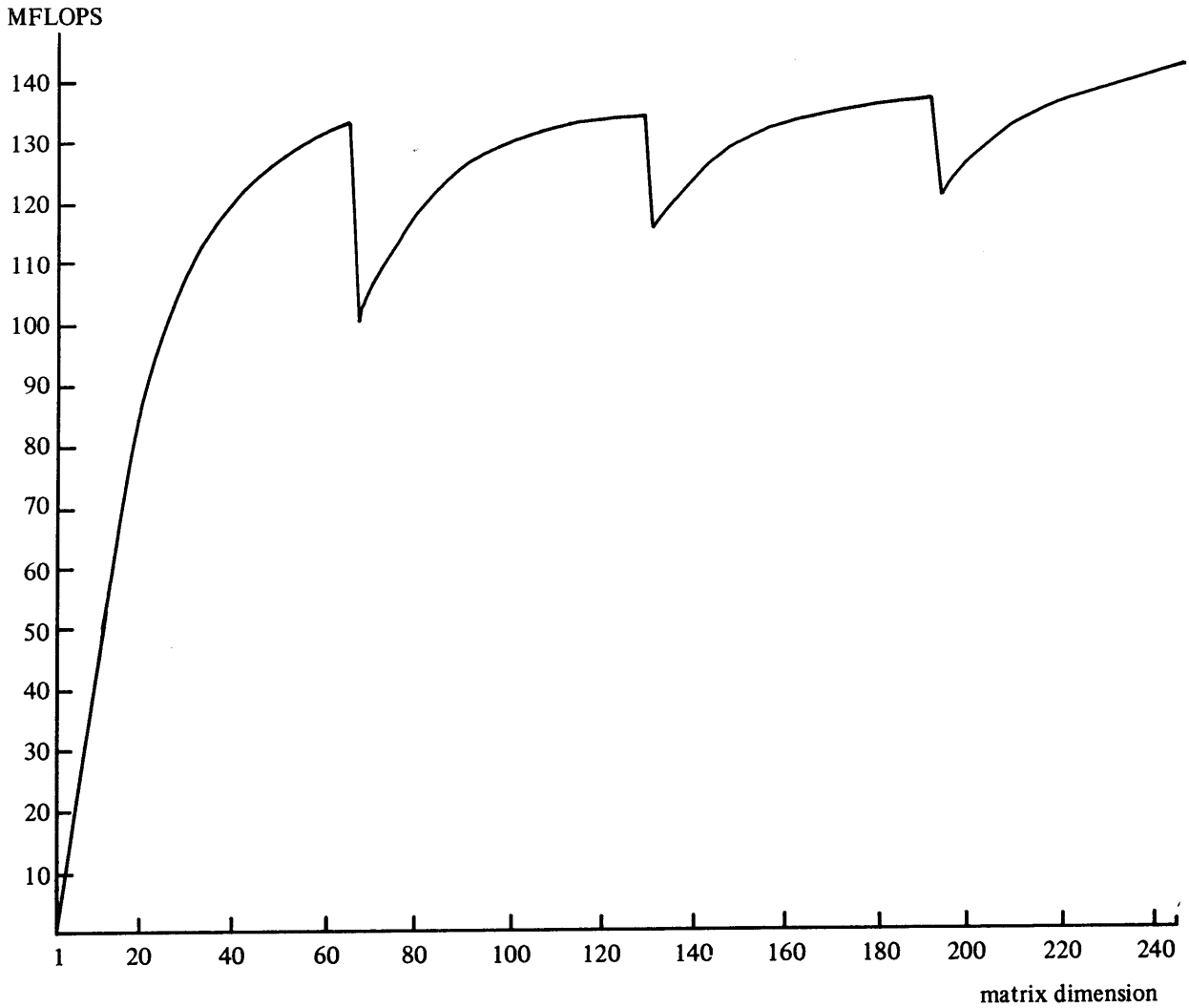


Figure 6. Matrix multiplication timing

Matrix multiplication is typical of the large class of problems which can be vectorized. For these problems a significant increase in processing speed can be achieved over conventional scalar processing. Register-to-register vector instructions and the large amount of concurrency attainable through use of the twelve independent functional units and chaining can provide processing speeds exceeding those of other existing vector machines. Fields such as weather forecasting, nuclear research and seismic analysis will find the CRAY-1 a welcome extension to their workshops of computing tools.

CRAY RESEARCH, INC.

PRELIMINARY REFERENCE MANUAL

CRAY-1 COMPUTER

THE CRAY-1 COMPUTER

PRELIMINARY

REFERENCE MANUAL

COPYRIGHT (C) 1975 BY CRAY RESEARCH, INC. THIS MANUAL, OR PARTS THEREOF, MAY NOT BE REPRODUCED IN ANY FORM WITHOUT PERMISSION OF CRAY RESEARCH, INC. THIS MANUAL CONTAINS PRELIMINARY INFORMATION WHICH IS SUBJECT TO CORRECTION AND CHANGE WITHOUT FORMAL NOTICE TO COPY HOLDERS.

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

REVISION RECORD

REVISION	DESCRIPTION
4/75	Original Printing
A	This revision corrects various typographical and technical
6/75	errors.

CONTENTS

Introduction 1

Summary of machine characteristics 1

Principal operating registers 1

A registers 1
B registers 1
S registers 2
T registers 2
V registers 2
VL register 2
VM register 2
P register 2

Supporting registers 5

BA register 5
LA register 5
XA register 5
F register 5
M register 6

Input/Output 6

Comparison of scalar and vector processing 6

Functional units 7

Instruction formats 7

Special register values 9

Instruction buffers. 9

Data formats 9

CAL language structure 11

Statement format 11

Location field 11
Result field 11
Operand field 11
Comments field 11

Coding conventions 12

Comments statement 12

Lower case in comments 12

Symbols 12

Special element, * 13

"P." prefix 13

"W." prefix 13

Expressions 13

CPU register designation 14

Pseudo-instructions 14

ABS 14
BSS 14
BSSZ 15
CON 15
EJECT 15
END 15
ENTRY 16
= 16
IDENT 16
LIST 16
ORG 17

CPU instructions 18

Appendix A. Summary of CPU instructions 60

Appendix B. Instruction timing 66

Appendix C. Coding examples 69

Appendix D. Use of the NOVA CAL assembler 71

Appendix E. Assembly errors 72

Appendix F. Description of binary output 73

FIGURES

Figure 1. Registers block diagram 3

Figure 2. Exchange package 4

Figure 3. Instruction formats 8

Figure 4. Data formats 10

Introduction

The Cray Research CRAY-1 computer is a powerful general purpose computer incorporating vector capabilities and a large, fast bi-polar memory. Vector processing provides result rates greatly exceeding the result rates of conventional scalar processing. The benefits of vector processing are visible even for short vectors. This manual introduces the characteristics of the CRAY-1 and describes the CRAY-1 assembly language (CAL).

Summary of machine characteristics

- 64-bit word
- 2's complement arithmetic
- scalar and vector processing modes
- 12 fully-segmented functional units
- eight 24-bit A registers
- sixty-four 24-bit B registers
- eight 64-bit S registers
- sixty-four 64-bit T registers
- eight 64-element V registers, 64 bits per element
- 4 instruction buffers of 64 parcels each
- 12.5 nanosecond clock period
- 1,048,576 words of bi-polar memory (64 bits and one parity bit) arranged in 16 banks
- 4 clock period bank cycle time
- 1 word/clock period transfer rate to B, T and V registers
- 1 word/two clock periods transfer rate to A and S registers
- 4 words/clock period transfer rate to instruction parcel buffers
- 12 full-duplex I/O channels

Principal operating registers

A registers

The eight 24-bit A registers are primarily used as address registers for memory references and as index registers. They are individually designated by the symbols A0, A1, A2, A3, A4, A5, A6 and A7. Data flows between these registers and the B, S and VL registers. Data may be directly transferred between the A registers and memory.

B registers

The sixty-four 24-bit B registers provide rapid-access temporary storage for the A registers. They are individually designated by the symbols B0, B1, B2, ..., B77. Data may be directly transferred between the B registers and memory.

S registers

The eight 64-bit S registers are the principal scalar registers for the CPU. They are individually designated by the symbols S0, S1, S2, S3, S4, S5, S6 and S7. These registers serve as source and designation registers in scalar arithmetic and logical instructions. They may also furnish one operand in vector instructions. Data flows between these registers and the A, T, V, and VM registers. Data may be directly transferred between the S registers and memory.

T registers

The sixty-four 64-bit T registers provide rapid-access temporary storage for the S registers. They are individually designated by the symbols T0, T1, T2, . . .,T77. Data may be directly transferred between the T registers and memory.

V registers

The eight 64-element V registers are the operating registers for vector computations. Each element is 64 bits. The V registers are individually designated by the symbols V0, V1, V2, V3, V4, V5, V6 and V7. These registers serve as source and destination registers in vector arithmetic and logical instructions. Data flows between these registers and the S registers. Data may be directly transferred between the V registers and memory.

VL register

The 7-bit VL register specifies the vector length. Vector computations are performed on vectors of the length specified by the contents of VL.

VM register

The 64-bit VM register contains a vector mask to control register selection in the vector merge instructions (146-147). Each bit of the VM register corresponds to a vector element.

P register

The 24-bit P register specifies the parcel address of the current program instruction. The high order 22 bits specify a memory address and the low order 2 bits specify the parcel number.

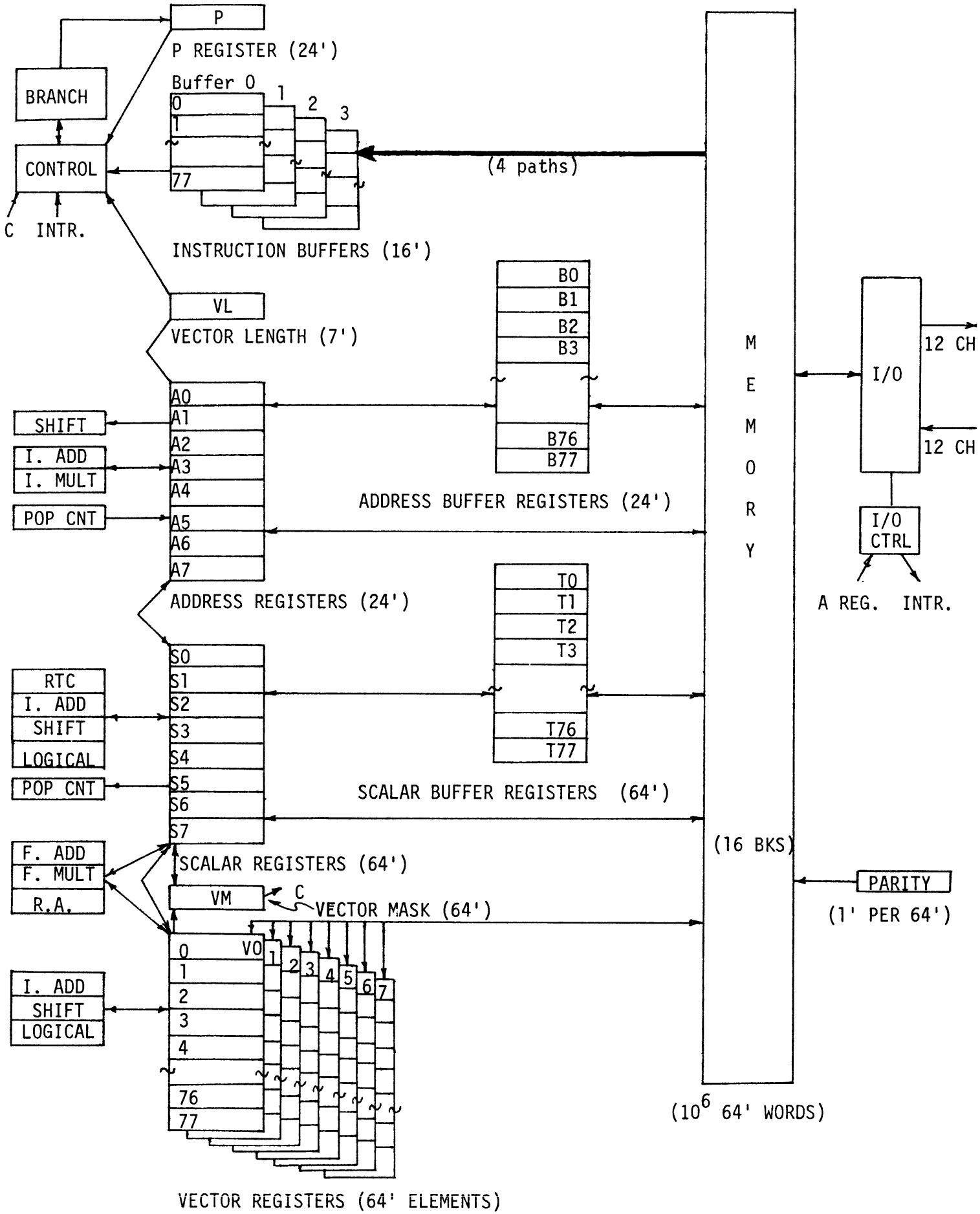
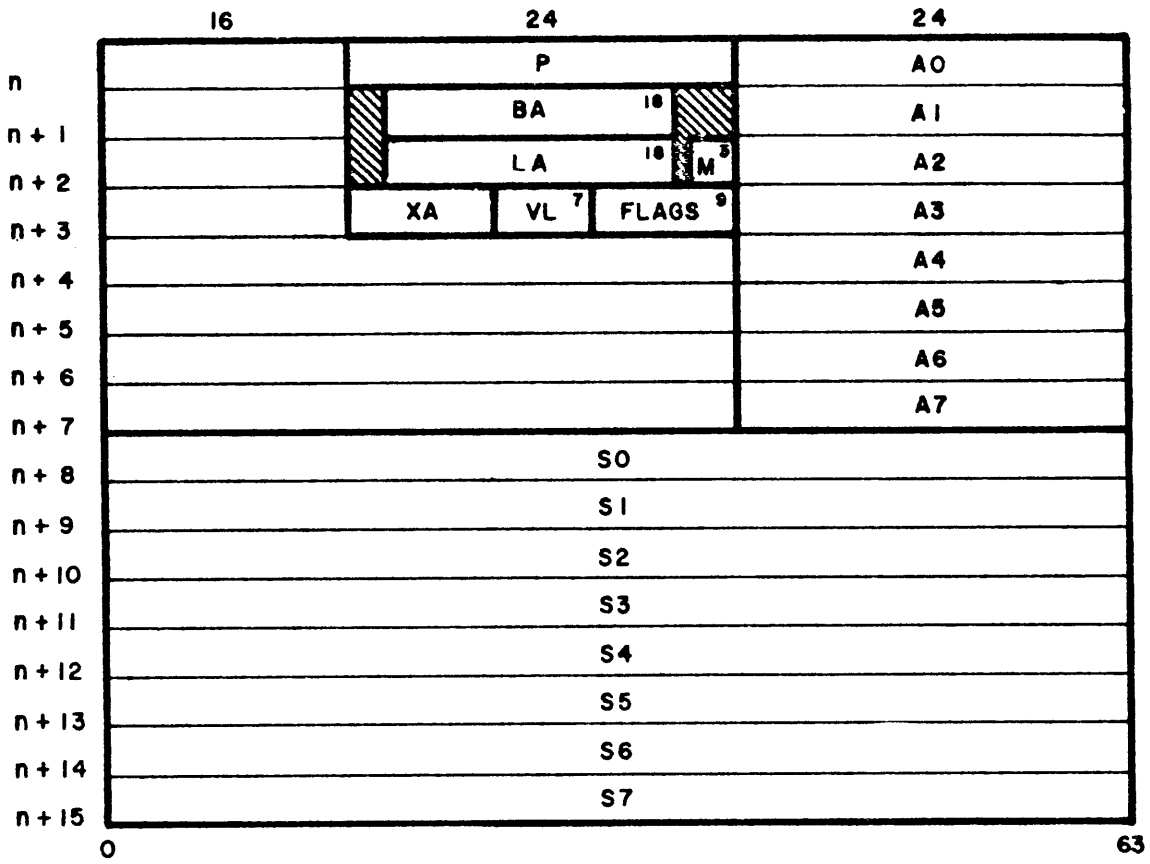


Figure 1. Registers block diagram



FLAGS

1. NORMAL EXIT
2. ERROR EXIT
3. I/O INTERRUPT
4. STORAGE PARITY
5. PROGRAM RANGE
6. OPERAND RANGE
7. FLOATING POINT ERROR (SET ON SCALAR REFERENCE ONLY)
8. RTC INTERRUPT
9. CONSOLE INTERRUPT

MODES

1. MONITOR MODE
2. INTERRUPT ON STORAGE PARITY
3. INTERRUPT ON FLOATING POINT

P = PROGRAM ADDRESS
BA = BASE ADDRESS
LA = LIMIT ADDRESS
XA = EXCHANGE ADDRESS
VL = VECTOR LENGTH

Figure 2. Exchange package

Supporting registers

The CPU contains a number of registers which support the operating registers in the execution of programs. These registers are loaded with new information during the execution of an exchange sequence. The information is not altered during the execution interval for an exchange package. These registers are listed below with a description of the individual function performed.

BA register

This 18-bit register holds the base address during the execution interval for each exchange package. The contents of this register is interpretted as the upper 18 bits of a 22-bit memory address. The lower 4 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.

LA register

This 18-bit register holds the limit address during the execution interval for each exchange package. The contents of this register is interpretted as the upper 18 bits of a 22-bit memory address. The lower 4 bits of the address are assumed zero. The BA and LA registers together provide memory protection. No memory references may be made below BA nor at or above LA. Such a reference will cause the program or operand range flag to be set and the execution interval of the exchange package will be terminated.

XA register

This 8-bit register holds the upper eight bits of a 12-bit exchange address during the execution interval for each exchange package. The low order 4 bits of the exchange address are assumed zero.

When the execution interval terminates, the exchange operation exchanges the contents of the registers with the contents of the exchange package at (XA)*16 in memory. The exchange operation saves the contents of the A, S, P, and VL registers and the supporting registers BA, LA, XA, M and F.

F register

This 9-bit register contains flags which are set to indicate the conditions causing an exchange operation. The interrupt conditions are:

- Normal exit
- Error exit
- I/O interrupt
- Storage parity
- Program range
- Operand range
- Floating point overflow (scalar only)
- External clock interrupt
- Console interrupt

M register

This 3-bit register specifies the modes for generation of interrupts. All interrupts are inhibited when the monitor mode bit is set. Interrupts on storage parity errors are enabled whenthe storage parity mode bit is set. Interrupts on scalar floating point overflow are enabled when the floating point mode bit is set.

Input/Output

There are twenty-four I/O channels, of which twelve are input channels and twelve are output channels. The channels are assigned the numbers 2 through 25. The channels are divided into four groups as follows:

Group 1	Input channels 2, 6, 10, 14, 18, 22
Group 2	Output channels 3, 7, 11, 15, 19, 23
Group 3	Input channels 4, 8, 12, 16, 20, 24
Group 4	Output channels 5, 9, 13, 17, 21, 25

Each input channel consists of a data channel (16 data bits and 3 control bits), a 64-bit assembly register, a current address (CA) register, and a channel limit address (CL) register. Each input channel can cause a CPU interrupt condition when the current address equals the limit address register value or when the input device sends a disconnect.

Each output channel consists of a data channel (16 data bits and 3 control bits), a 64-bit disassembly register, a current address (CA) register, and a channel limit address (CL) register. Each output channel can cause a CPU interrupt condition when the current address equals the limit address register value. A disconnect is sent on the output channel after the last word of a record is sent and acknowledged.

Comparison of scalar and vector processing

Scalar instructions apply a function to one or two operands in registers and enter the result into a register. The addition of two integers in S1 and S2, entering the sum into S3, is an example of a scalar instruction. Vector instructions apply a function to sets of operands called vectors. Suppose one wanted to perform several additions like the one above. One could execute a small loop which would perform one addition per pass, saving S3 sums as they are generated. Alternatively, one could enter the addends into elements of one V register and the augends into elements of another V register and then execute a single vector addition instruction. The set of addends, the set of augends, and the set of sums are vectors. Vector processing provides much higher result rates than can be obtained by conventional scalar processing.

Functional units

There are twelve functional units in the computation section of the CPU. Each is a specialized unit implementing algorithms for a portion of the instructions. Each unit is independent of the other units and a number of functional units may 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. There is no information retained in a functional unit for reference in subsequent instructions. These units operate essentially in three-address mode with very limited source and destination addressing.

Three functional units provide 24-bit results to the A registers only:

- integer add
- integer multiply
- population count

Three functional units provide 64-bit results to the S registers only:

- integer add
- shift
- logical

Three functional units provide 64-bit results to the V registers only:

- integer add
- shift
- logical

Three functional units provide 64-bit results to either the S or V registers:

- floating add
- floating multiply
- reciprocal approximation

All functional units have one clock period segmentation. This means that the information arriving at the unit, or moving within the unit, is captured and held in a new set of registers at the end of every clock period. It is therefore possible to start a new set of operands for unrelated computation into a functional unit each clock period even though the unit may require more than one clock period to complete the calculation. 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. Functional units servicing the vector instructions produce one result per clock period.

Instruction formats

Figure 3 illustrates the five instruction formats for the CRAY-1. Each instruction is either a one-parcel (16-bit) instruction or a two-parcel (32-bit) instruction. Two-parcel instructions may begin in the fourth and last parcel position within a word and end in the first parcel position of the next word. The assembler lists a parcel address as a word address followed by a one-character alphabetic (a-d) parcel identifier.

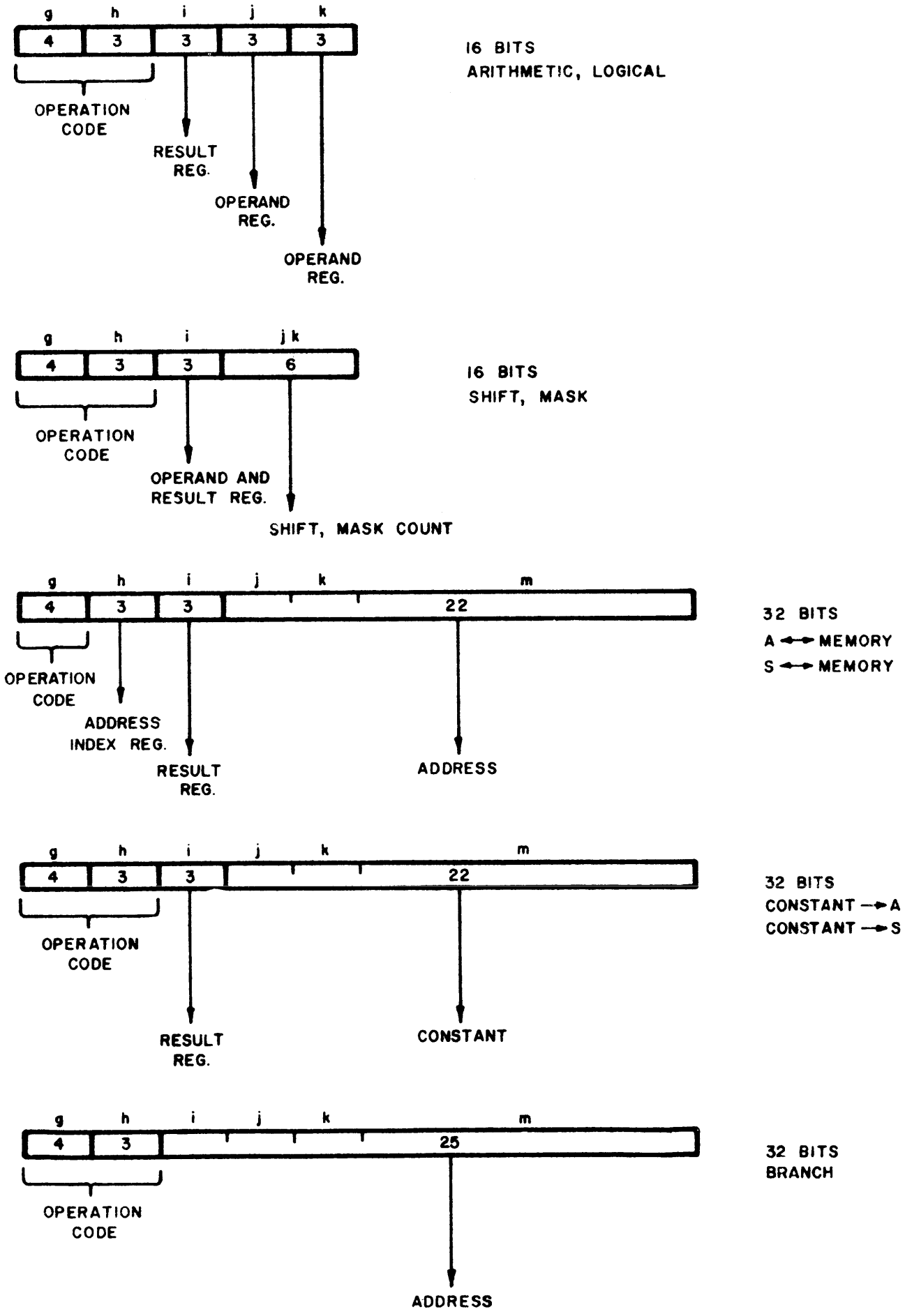


Figure 3. Instruction formats

Special register values

S0 and A0 provide special values when they are designated in the j or k portions of an instruction. In these cases the special value is used as the operand and the contents of S0 or A0 is ignored. If an S0 or A0 operand is designated in the i portion of an instruction, the actual contents of S0 or A0 is used as the operand. The instruction descriptions enumerate the uses of the special register values where they are meaningful.

register	value
Ai,i=0	A0
Aj,j=0	0
Ak,k=0	1
Si,i=0	S0
Sj,j=0	0
Sk,k=0	2^{63}

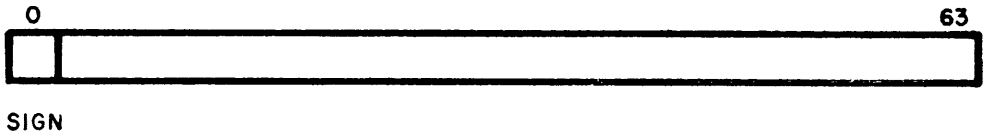
Instruction buffers

There are four instruction buffers each consisting of sixty-four 16-bit registers. All instructions are executed from the instruction buffers. An instruction buffer supplies instructions to the next instruction parcel (NIP) and the current instruction parcel (CIP) registers. Associated with each instruction buffer is a base address register that specifies the high order 18 bits of the parcel addresses contained in the instruction buffer. The base address registers are scanned each clock period. If the high order 18 bits of the P register matches one of the base addresses, the proper instruction is selected from the instruction buffer and sent to the NIP register. The instruction is moved to the CIP register for execution. The second parcel of a 2-parcel instruction resides in the NIP register when the instruction issues.

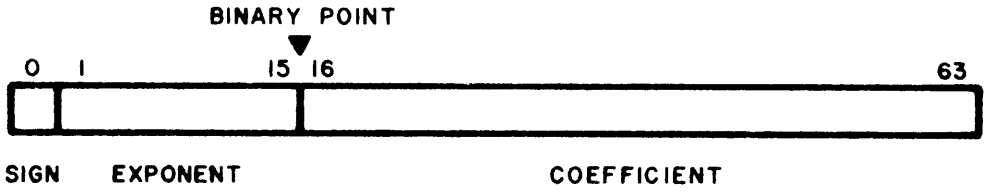
When the high order 18 bits of the P register do not match any instruction buffer base address, an "out of buffer" condition exists and instructions are read to an instruction buffer from memory. When an "out of buffer" condition occurs, the instruction buffer that receives the instruction is determined by a 2-bit counter. Each occurrence of an "out of buffer" condition causes the counter to be incremented. The first four instruction parcels in an instruction buffer are always from bank 0, however, the first parcels read into an instruction buffer always include the parcel specified by the contents of the P register.

Data formats

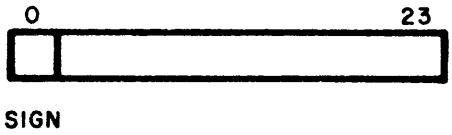
Figure 4 illustrates the data formats for integers and floating point quantities. The range for floating point quantities is $\left[10^{-5000}, 10^{2500}\right]$. Normalized floating point quantities are expressed as $z = y * 2^x$ where $y = 0$ or $\frac{1}{2} \leq y < 1$ and $-40000_8 \leq x < 20000_8$. The exponent of x is expressed in excess-40000_8 notation. The exponent of a floating point quantity is obtained by adding 40000_8 to the true exponent. Overflow is indicated by an exponent exceeding 57777_8.



2's COMPLEMENT INTEGER (64 BITS)



SIGNED MAGNITUDE FLOATING POINT (64 BITS)



2's COMPLEMENT INTEGER (24 BIT)

Figure 4. Data formats

CAL language structure

Statement format

A CAL language source program consists of a sequence of symbolic machine instructions,pseudo instructions and comment lines. Except for comment lines, each statement consists of a location field, a result field, an operand field, and a comments field. Each field is terminated by one or more blank characters. Statement format is essentially free field.

Statements are 80 column lines. When punched on cards, each card is considered a line. Information beyond column 72 is not interpretted by CAL but does appear on the assembly listing. Thus, columns 73-80 can be used for additional comments or sequencing.

Location field

The location field entry begins in column one or two of a new statement line and is terminated by a blank. If columns one and two are blank, the location field has no entry.

Result field

If the location field is blank, the result field can begin in column three. If the location field is nonblank, the result field begins with the first nonblank character following the location field and is terminated by one or more blanks. The result field is blank if there are no nonblank characters between the location field and column 35.

Operand field

The contents of the result field determinesif any entry is required in the operand field. The operand field begins with the first nonblank character following the result field and is terminated by one or more blanks. It is blank if there are no nonblank characters between the result field and column 35.

Comments field

Comments are optional and begin with the first nonblank character following the operand field, or, if the operand field is missing, begin no earlier than column 35.

Coding conventions

The following coding convention should be adopted to assure uniformity of all CRAY-1 systems code:

Column	Contents
1	Asterisk (comments statement only)
1-8	Location field entry, left-justified
9	Blank
10-18	Result field entry, left-justified
19	Blank
20-34	Operand field entry, left-justified
35	Beginning of comments

Comments statement

A comments statement is designated by either an asterisk in column 1 or by blanks in columns 1-34. Comments statements are listed in assembler output but have no other effect on assembly.

Lower case in comments

Since the standard keypunch requires multipunching of lower case characters, an escape character is provided to indicate that succeeding alphabetic characters (A-Z) are to be converted to lower case. The conversion is performed only for comments statements and comments fields. Conversion is terminated by a subsequent occurrence of the escape character, which may be on a different card, or by the end of the program. When in lower case mode, a single alphabetic character may be capitalized by prefixing the capitalization character. The capitalization character has no effect if not followed by an alphabetic character.

	character	card code	ASCII code
Escape character	SUB	7-8-9	032
Capitalization character	EOT	7-9	004

Symbols

A symbol is a string of 1-8 characters that defines a value and its associated attributes. The first character must be alphabetic (A-Z), @ or $. Second and successive characters may also be digits (0-9) or =. A symbol may have a word address or a parcel address attribute, or neither of these. A symbol is a parcel address if it appears in the location field of an instruction. A symbol is a word address if it appears in the location field of a CON, BSS, or BSSZ pseudo instruction. The "=" statement can be used to define a symbol with either attribute.

Special element, *

The use of the special element * in an expression causes the assembler to replace it with the current value of the location counter.

"P." prefix

A symbol or constant may be prefixed by a "P." to specify the attribute of parcel address. If a symbol, sym, has the attribute of word address, the value of P.sym is the value of sym multiplied by four. A "P." prefix to a symbol with neither word nor parcel address attributes or to a constant does not cause the value to be multiplied by four, but it can be used to assign the parcel address attribute to a symbol being defined by an "=" statement.

"W." prefix

A symbol or constant may be prefixed by "W." to specify the attribute of word address. If a symbol, sym, has the attribute of parcel address, the value of W.sym is the value of sym divided by four. A "W." prefix to a symbol with neither word nor parcel address attributes or to a constant does not cause the value to be divided by four, but it can be used to assign the word address attribute to a symbol being defined by an "=" statement.

Expressions

Expressions are evaluated from left to right without regard for operator (+, -, * and /) precedence. Expressions in branch instructions (006-017) must not evaluate to type "word address". Expression elements may be one of the following forms:

```
*
symbol
octal constant
O'nnnn    (nnnn, an octal constant)
D'nnnn    (nnnn, a decimal constant)
A'cccc'   (cccc, a character string)
'cccc'
```

A'cccc' or 'cccc' left-justifies the character string in a 64-bit field with blank fill. A suffix may be used to specify an alternate justification or fill: H - same as no suffix, L - left-justified, zero fill, or R - right-justified, zero fill. An apostrophe in a character string is represented as two apostrophes. A null expression is given the value zero.

CPU register designation

A, S, and V registers are designated by suffixing a single octal digit, n (An, Sn, Vn). B and T registers are designated by suffixing one or two octal digits (Bnn, Tnn). A symbol may be used in place of a B or T register number if the register name and number are separated by a period. The symbol must have been previously defined. For example,

```
RTNADDR  =  14
         J  B.RTNADDR
```

accomplishes the same thing as

```
         J  B14  .
```

Pseudo instructions

Three pseudo instructions are required for an assembly: IDENT, ENTRY, and END. IDENT must be the first source statement. END signals the termination of source statements for a program. Statements preceding the first IDENT or between a succeeding END and subsequent IDENT are interpretted as comments.

ABS

The ABS pseudo instruction specifies that the program is absolute. This pseudo must precede any BSS, BSSZ, CON, = or instruction. It has no real purpose for the NOVA CAL assembler and may be omitted, but it is implemented to facilitate the eventual relocatable module capability.

Example:

location	result	operand	comments
1	10	20	35
	ABS		

BSS

The BSS pseudo instruction causes a block of storage to be reserved. The location counter is first rounded to the next word boundary (force upper), and then the number of words specified by the operand field expression is reserved. Unused parcels are padded with pass instructions (S1 S1&S1). A location field symbol, if present, is assigned the value of the current word address after the force upper occurs.

Example:

location	result	operand	comments
1	10	20	35
	BSS	W.*-W.100	

BSSZ

The BSSZ pseudo instruction causes a block of zero storage to be reserved. The location counter is first rounded to the next word boundary (force upper), and then the number of zero words specified by the operand field expression is reserved. A location field symbol, if present, is assigned the value of the current word address after the force upper occurs.

Example:

location	result	operand	comments
1	10	20	35
	BSSZ	177	

CON

The CON pseudo instruction generates a full word of binary data. This pseudo always forces upper. A location field symbol, if present, is assigned the value of the current word address after the force upper occurs.

Example:

location	result	operand	comments
1	10	20	35
	CON	7777017	

EJECT

The EJECT pseudo instruction causes the next listing line to appear on a new page. The EJECT pseudo itself is not listed. The EJECT pseudo has no effect when the E global switch is selected.

Example:

location	result	operand	comments
1	10	20	35
	EJECT		

END

The END pseudo instruction indicates the end of the program. Subsequent cards, if any, are assumed to be part of the next program.

Example:

location	result	operand	comments
1	10	20	35
	END		

ENTRY

The ENTRY pseudo instruction specifies an entry point of the program. Only one ENTRY pseudo is permitted by the initial CAL assembler. The entry point name is specified in the operand field and must subsequently appear in the location field of an instruction or "=" pseudo instruction.

Example:

location	result	operand	comments
1	10	20	35
	ENTRY	EPTNME	

=

The "=" pseudo instruction defines the symbol in the location field as having the value and attribute indicated by the expression in the operand field. Any symbol in the expression must be previously defined. If the expression is erroneous, CAL does not define the location symbol but flags an error.

Example:

location	result	operand	comments
1	10	20	35
SYMB	=	A*B+100/4	

IDENT

The name of the program is specified in the operand field of the IDENT pseudo instruction. The name must be 1-8 characters, of which the first must be alphabetic (A-Z), @ or $. Second and successive characters may also be digits (0-9) or =. The name appears in the listing heading and in the program descriptor table (PDT) of the absolute module.

Example:

location	result	operand	comments
1	10	20	35
	IDENT	PMJ	

LIST

The LIST pseudo instruction controls the listing. If the operand field is empty the listing is suppressed until encountering another LIST pseudo with a non-empty operand field, or until the end of the program.

Example:

location	result	operand	comments
1	10	20	35
	LIST	ON	

ORG

The ORG pseudo instruction specifies the origin of the program. This pseudo must precede any BSS, BSSZ, CON, = or instruction. The origin is specified in the operand field. If omitted, an origin of zero is assumed.

Example:

location	result	operand	comments
1	10	20	35
	ORG	0'100	

- 000 ERR Error Exit

This instruction is treated as an error condition and an exchange jump occurs. The contents of the instruction buffers are voided by the exchange jump. If not in monitor mode, the error 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 jump 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 jump is advanced one count from the address of the error exit instruction. The error exit instruction is not intended for use in user program code. Its purpose is to halt execution of an incorrectly coded program which jumps into an unused area of memory or into a data area.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
000000		ERR		

- 001 CA,Aj Ak Set the channel (Aj) current address to (Ak) and begin the I/O sequence
  CL,Aj Ak Set the channel (Aj) limit address to (Ak)
  CI,Aj Clear the channel (Aj) interrupt flag and error flag
  XA Aj Enter the XA register with (Aj)
  RT Sj Enter the real time clock register with (Sj)

This instruction performs specialized functions useful to the operating system. The instruction is treated as a pass instruction if not in monitor mode or if the i designator is 5, 6 or 7.

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 contains the address of the current channel word. The LA register specifies the limit address. In programming the channel, the LA register is initialized and setting CA activates the channel. As the transfer continues CA is incremented toward LA. When CA=LA the transfer is complete. When the j designator is 0 or when the contents of Aj is less than 2 or greater than 25, these functions are executed as pass instructions. When the k designator is 0, CA or LA is set to 1.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
001035		CA,A3	A5	
001134		CL,A3	A4	
001210		CI,A1		

When the i designator is 3, the instruction causes the exchange address (XA) register to be set to the contents of Aj. When the j designator is zero, the XA register is cleared. A monitor program activates a user job by initializing the XA register with the address of the user job's exchange package and then executing a normal exit (004).

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
001350		XA	A5	
001420		RT	S2	
001400		RT	S0	clear RTC

- 002 VL Ak 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. Vector instructions operate on vectors whose lengths are determined by subtracting one from the contents of VL; one plus the contents of the low order six bits of the result is the vector length. The maximum vector length of 64 can be achieved by setting the contents of Ak to zero or 64 before executing this instruction. When the k designator is zero, the vector length is set to one.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
020200000100		A2	D'64	VL = 64
002002		VL	A2	
022100		A1	0	VL = 64
002001		VL	A1	

- 003 VM Sj Transmit Sj to VM

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) where an operation is performed depending on the contents of VM.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
003040		VM	S4	
003000		VM	S0	VM = 0

- 004 EX Normal exit

This instruction causes an exchange jump. The contents of the instruction buffers are voided by the exchange jump. If not in monitor mode, 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 jump 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. The value of an optional operand field expression is inserted in the lower 9 bits of the instruction. All symbols in the expression must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
004000		EX		
004027		EX	27	

- 005 J Bjk Branch to (Bjk)

This instruction sets the P register to the parcel address specified by the contents of Bjk, and execution continues at that address.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
005017		J	B17	
005017		J	B.RTNADDR	(RTNADDR=17)

- 006 J exp Branch to ijkm

This instruction sets the P register to the parcel address specified by the low order 24 bits of the ijkm field, and execution continues at that address. The high order bit of the ijkm field is ignored.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
006 00002124b		J	TAG1	
006 00001752d		J	LDY3+1	
006 00004530a		J	*+3	

- 007 R exp Return jump to ijkm

This instruction sets register B00 to the address of the following parcel. The P register is then set to the parcel address specified by the low order 24 bits of the ijkm field, and 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 instruction on the contents of B00 (005).

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
007 00001142d		R	HELP	

- 010 JAZ exp Branch to ijkm if A0=0
- 011 JAN exp Branch to ijkm if A0≠0
- 012 JAP exp Branch to ijkm if A0 positive
- 013 JAM exp Branch to ijkm if A0 negative

These instructions tests the contents of A0 for the condition specified. If the condition is satisfied, the P register is set to the parcel address specified by the low order 24 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. If A0 contains zero, it is considered positive.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
010 00002245b		JAZ	TAG3+2	
011 00004520a		JAN	P.CON1	
012 00002221c		JAP	TAG2	
013 00002124b		JAM	TAG1	

- 014 JSZ exp Branch to ijkm if S0=0
- 015 JSN exp Branch to ijkm if S0≠0
- 016 JSP exp Branch to ijkm if S0 positive
- 017 JSM exp Branch to ijkm if S0 negative

These instructions tests the contents of S0 for the condition specified. If the condition is satisfied, the P register is set to the parcel address specified by the low order 24 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. If S0 contains zero, it is considered positive.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
014 00002221c		JSZ	TAG2	
015 00002124d		JSN	TAG1+2	
016 00004540d		JSP	*+3	
017 00002367c		JSM	TAG4	

- 020 Ai exp Transmit jkm to Ai
  Ai #exp

This two-parcel instruction enters the 22-bit quantity of the jkm field into Ai. The quantity is treated as a 22-bit positive integer; the upper bits of Ai are cleared. The assembler generates this instruction when no # symbol precedes the expression and the value of the expression exceeds 63. If all symbols in the expression have not been previously defined, this instruction is generated when the expression value is positive even though the value may be less than 64. When the # symbol precedes the expression, the expression is first evaluated and if the value is negative, it is complemented and the complemented value is stored in the jkm field. The complement is formed by changing all 1 bits to zero and all 0 bits to one. When the expression is positive, an 021 instruction is generated (see below).

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
020200000130		A2	130	
020300000021		A3	VAL+1	
020401777777		A4	1777777	
020500051531		A5	A'SY'R	
020600000000		A6	#MINUS1	MINUS1=-1

- 021 Ai exp Transmit jkm to Ai and complement
Ai #exp

This two-parcel instruction enters the 22-bit quantity of the jkm field into Ai and complements the result. The complement is formed by changing all 1 bits to zero and all 0 bits to one. This instruction is used to enter a negative number into an A register. The assembler generates this instruction when the value of the expression is negative and no # symbol precedes the expression; the jkm field will contain the complement of the expression. When the # symbol precedes the expression, the expression is first evaluated and if the value is positive it is stored in the jkm field. When the expression is negative, an 020 instruction is generated (see above).

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
021200000010		A2	#10	
021200000007		A2	-10	

- 022 Ai exp Transmit jk to Ai

This one-parcel instruction enters the 6-bit quantity of the jk field into Ai. All symbols in the expression must be previously defined. If all symbols are not previously defined, an 020 instruction is generated even though the value may be less than 64.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
022310		A3	10	

- 023 Ai Sj Transmit Sj to Ai

This instruction enters the low order 24 bits of Sj into Ai. Ai is cleared if the j designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
023410		A4	S1	

- 024 Ai Bjk Transmit Bjk to Ai

This instruction enters the contents of Bjk into Ai. A symbolic B register number must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
024517		A5	B17	
024517		A5	B.SVNTEEN	

- 025 Bjk Ai Transmit Ai to Bjk

This instruction enters the contents of Ai into Bjk. A symbolic B register number must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
025634		B34	A6	
025634		B.THRTY4	A6	

- 026 Ai PSj Population count of Sj to Ai

This instruction counts the number of one bits in Sj and enters the result into Ai. Ai is cleared if the j designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
026720		A7	PS2	

- 027 Ai ZSj Leading zero count of Sj to Ai

This instruction counts the number of leading zeroes in Sj and enters the result into Ai. Ai is set to 64 if the j designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
027130		A1	ZS3	

- 030 Ai Aj+Ak Integer sum of Aj and Ak to Ai

This instruction forms the integer sum of Aj and Ak and enters the result into Ai. No overflow is detected. Ak is transmitted to Ai when the j designator is zero and the k designator is non-zero. One is transmitted to Ai when the j and k designators are both zero. (Aj)+1 is transmitted to Ai when the j designator is non-zero and the k designator is zero. The assembler allows alternate forms for this instruction when either j or k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
030123		A1	A2+A3	
030102		A1	A2	(special form)
030230		A2	A3+1	(special form)

- 031 Ai Aj-Ak Integer difference of Aj and Ak to Ai

This instruction forms the integer difference of Aj and Ak and enters the result into Ai. No overflow is detected. The negative of Ak is transmitted to Ai when the j designator is zero and the k designator is non-zero. -1 is transmitted to Ai when the j and k designators are both zero. (Aj)-1 is transmitted to Ai when the j designator is non-zero and the k designator is zero. The assembler allows alternate forms for this instruction when either j or k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
031456		A4	A5-A6	
031102		A1	-A2	(special form)
031450		A4	A5-1	(special form)

- 032 Ai Aj*Ak 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. Ai is cleared when the j designator is zero. Aj is transmitted to Ai when the k designator is zero and the j designator is non-zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
032712		A7	A1*A2	

- 033 Ai CI Channel number of highest priority interrupt request to Ai
Ai CA,Aj Current address of channel (Aj) to Ai
Ai CE,Aj Error flag of channel (Aj) to Ai

This instruction enters channel status information into Ai. The j and k designators and the contents of Aj define the desired information. 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 non-zero. The value of the current address (CA) register for the channel is entered into Ai when the k designator is zero. The error flag for the channel is entered into the low order bit of Ai and the error flag is cleared when the k designator is one. The high order bits of Ai are cleared.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
033100		A1	CI	
033230		A2	CA,A3	
033341		A3	CE,A4	

- 034 B,Ai,exp ,A0 Read jk+1 words starting at B register (Ai) from memory starting at (A0)

This instruction is used to read the low order 24 bits of words from memory directly into the B registers. A0 contains the address in memory of the first word. The B register which is to receive the first word is specified by the contents of Ai. Subsequent words are stored in consecutive B registers. Processing of the B registers is circular. B00 is processed after B77 if the count is not exhausted. Symbols in the expression must be previously defined. The expression must have a positive value between 1 and 64 and is a true count of the number of words to be read. The assembler subtracts one from the value of the expression and stores the result in the jk field of the instruction. A0 in the operand field is optional.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
034407		B,A4,10	,A0	
034407		B,A4,10	,	
034516		B,A5,VAL+1	,A0	

- 035 ,A0 B,Ai,exp Store jk+1 words starting at B register (Ai) to memory starting at (A0)

This instruction is used to store the B registers directly into memory. A0 contains the address in memory to receive the first word. The B register which is stored at the first address is specified in Ai. Subsequent B registers are stored in consecutive words in memory. Processing of the B registers is circular. B00 is processed after B77 if the count is not exhausted. Symbols in the expression must be previously defined. The expression must have a positive value between 1 and 64 and is a true count of the number of words to be stored. The assembler subtracts one from the value of the expression and stores the result in the jk field of the instruction. A0 in the result field is optional.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
035522		,A0	B,A5,23	
035522		,	B,A5,23	
035516		,A0	B,A5,VAL+1	

- 036 T,Ai,exp ,A0 Read jk+1 words starting at T register (Ai) from memory starting at (A0)

This instruction is used to read 64-bit words from memory directly into the T registers. A0 contains the address in memory of the first word. The T register which is to receive the first word is specified by the contents of Ai. Subsequent words are stored in consecutive T registers. Processing of the T registers is circular. T00 is processed after T77 if the count is not exhausted. Symbols in the expression must be previously defined. The expression must have a positive value between 1 and 64 and is a true count of the number of words to be read. The assembler subtracts one from the value of the expression and stores the result in the jk field of the instruction. A0 in the operand field is optional.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
036407		T,A4,10	,A0	
036407		T,A4,10	,	
036516		T,A5,VAL+1	,A0	

- 037 ,A0 T,Ai,exp Store jk+1 words starting at T register (Ai) to memory starting at (A0)

This instruction is used to store the T registers directly into memory. A0 contains the address in memory to receive the first word. The T register which is stored at the first address is specified in Ai. Subsequent T registers are stored in consecutive words in memory. Processing of the T registers is circular. T00 is processed after T77 if the count is not exhausted. Symbols in the expression must be previously defined. The expression must have a positive value between 1 and 64 and is a true count of the number of words to be stored. The assembler subtracts one from the value of the expression and stores the result in the jk field of the instruction. A0 in the result field is optional.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
037522		,A0	T,A5,23	
037522		,	T,A5,23	
037516		,A0	T,A5,VAL+1	

- 040 Si exp Transmit jkm to Si
  Si #exp

This two-parcel instruction enters the 22-bit quantity of the jkm field into Si. The quantity is treated as a 22-bit positive integer; the upper bits of Si are cleared. The assembler generates this instruction when no # symbol precedes the expression and the value of the expression is positive. When the # symbol precedes the expression, the expression is first evaluated and if the value is negative, it is complemented and the complemented value is stored in the jkm field. The complement is formed by changing all 1 bits to zero and all 0 bits to one. When the expression is positive, an 041 instruction is generated (see below).

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
040200000130		S2	130	
040300000021		S3	VAL+1	
040401777777		S4	1777777	
040500051531		S5	A'SY'R	
040600000000		S6	#MINUS1	MINUS1=-1

- 041 Si exp Transmit jkm to Si and complement
Si #exp

This two-parcel instruction enters the 22-bit quantity of the jkm field into Si and complements the result. The complement is formed by changing all 1 bits to zero and all 0 bits to one. This instruction is used to enter a negative number into an S register. The assembler generates this instruction when the value of the expression is negative and no # symbol precedes the expression; the jkm field will contain the complement of the expression value. When the # symbol precedes the expression, the expression is first evaluated and if the value is positive, it is stored in the jkm field. When the expression is negative, an 040 instruction is generated (see above).

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
041200000000		S2	-1	
041300000002		S3	#2	
041401777776		S4	-1777777	
041400000003		S4	#VAL2	(VAL2=3)

- 042 Si <exp Form ones mask in Si from the right
Si #>exp

This instruction is used to generate a mask of ones from the right. The assembler evaluates the expression to determine the mask length. If the # symbol precedes the expression the mask length is 64 minus the expression value. All symbols in the expression must be previously defined. The assembler stores 64 minus the mask length in the jk field of the instruction. The mask length must be a positive integer not exceeding 64. If the mask length is zero, an 043 instruction is generated.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
042273		S2	<5	
042273		S2	#>73	
042366		S3	<D'10	
042400		S4	<100	
043500		S5	<0	
042677		S6	1	(special form)

- 043 Si >exp Form ones mask in Si from the left
Si #<exp

This instruction is used to generate a mask of ones from the left. The assembler evaluates the expression to determine the mask length. If the # symbol precedes the expression the mask length is 64 minus the expression value. All symbols in the expression must be previously defined. The assembler stores the mask length in the jk field of the instruction. The mask length must be a positive integer not exceeding 64. If the mask length is 64, an 042 instruction is generated.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
043205		S2	>5	
043205		S2	#<73	
043312		S3	>D'10	
042400		S4	>100	
043500		S5	>0	
043600		S6	0	

- 044 Si Sj&Sk Logical product of Sj and Sk to Si

This instruction forms the logical product (AND) of Sj and Sk and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of Sj and Sk are 1 as in the following example:

(Sj) = 1100
(Sk) = 1010
(Si) = 1000

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 non-zero and the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
044234		S2	S3&S5	
044655		S6	S5&S5	S5 to S6
044307		S3	S0&S7	clear S3
044160		S1	S6&S0	get sign of S6

- 045 Si #Sk&Sj Logical product of Sj and complement of Sk to Si

This 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 1 when the corresponding bits of Sj and the complement of Sk are 1 as in the following example:

(Sj) = 1100
(Sk) = 1010
(Si) = 0100

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.

Examples:

Code Generated

location	result	operand	comments
1	10	20	35
045271	S2	#S1&S7	
045433	S4	#S3&S3	clear S4
045506	S5	#S6&S0	clear S5
045670	S6	#S0&S7	clear sign bit

- 046 Si Sj\Sk Logical difference of Sj and Sk to Si

This instruction forms the logical difference (exclusive OR) of Sj and Sk and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of Sj and Sk are different as in the following example:

(Sj) = 1100
(Sk) = 1010
(Si) = 0110

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

Examples:

Code Generated

location	result	operand	comments
1	10	20	35
046123	S1	S2\S3	
046455	S4	S5\S5	clear S4
046506	S5	S0\S6	S6 to S5
046770	S7	S7\S0	toggle sign

- 047 Si #Sj\Sk Logical difference of Sk and Sj complement to Si

This instruction froms the logical difference (exclusive OR) of Sk and the complement of Sj and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of Sj and Sk are the same, as in the following example:

(Sj) = 1100
(Sk) = 1010
(Si) = 1001

Si is set to all ones if the j and k designators have the same non-zero value. The complement of Sk is transmitted to Si if the j designator is zero and the k designator is non-zero. The assembler allows a special form for this case, as shown in the example below. All bits except the sign bit of Sj are complemented and the result is transmitted to Si if the j designator is non-zero and the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
047345		S3	#S4\S5	
047607		S6	#S7	(special form)

- 050 Si Sj!Si&Sk Scalar merge

This instruction merges the contents of Sj with Si depending on the ones mask in Sk. The result is defined by (Sj&Sk)!(Si&#Sk) as in the following example:

(Sk) = 11110000
(Si) = 11001100
(Sj) = 10101010
(Si) = 10101100

This 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 1 if the j designator is zero and the k designator is non-zero. The sign bit of Sj replaces the sign bit of Si if the j designator is non-zero and the k designator is zero. The sign bit of Si is cleared if the j and k designators are both zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
050123		S1	S2!S1&S3	
050760		S7	S6!S7&S0	

- 051 Si Sj!Sk Logical sum of Sj and Sk to Si

This 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) = 1100
(Sk) = 1010
(Si) = 1110
```

Sj is transmitted to Si if the j and k designators have the same non-zero value. Sk is transmitted to Si if the j designator is zero and the k designator is non-zero. The assembler allows a special form for this case, as in the example below. Sj, with the sign bit set to 1, is transmitted to Si if the j designator is non-zero 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.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
051472		S4	S7!S2	
051366		S3	S6!S6	
051701		S7	S0!S1	
051701		S7	S1	(special form)

- 052 S0 Si<exp Shift Si left jk places to S0

This instruction shifts Si left jk places and enters the result into S0. The assembler evaluates the expression to determine the shift count. All symbols in the expression must be previously defined. The shift count must be a positive integer not exceeding 64. If the shift count is 64, an 053 instruction is generated. The shift is end-off with zero fill. Si is not altered.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
052305		S0	S3<5	
052012		S0	S0<D'10	
052760		S0	S7<VAL	
053200		S0	S2<100	

- 053 S0 Si>exp Shift Si right jk negative places to S0

This instruction shifts Si right (64-jk) places and enters the result into S0. The assembler evaluates the expression to determine the shift count. All symbols in the expression must have been previously defined. The shift count must be a positive integer not exceeding 64. The assembler stores 64 minus the shift count in the jk field of the instruction. If the shift count is zero, an 052 instruction is generated. The shift is end-off with zero fill. Si is not altered.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
053373		S0	S3>5	
053066		S0	S0>D'10	
053760		S0	S7>VAL	
052100		S0	S1>0	

- 054 Si Si<exp Shift Si left jk places

This instruction shifts Si left jk places and enters the result into Si. The assembler evaluates the expression to determine the shift count. All symbols in the expression must have been previously defined. The shift count must be a positive integer not exceeding 64. If the shift count is 64, an 055 instruction is generated. The shift is end-off with zero fill.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
054703		S7	S7<3	
054656		S6	S6<VAL+2	
055300		S3	S3<100	

- 055 Si Si>exp Shift Si right jk negative places

This instruction shifts Si right (64-jk) places and enters the result into Si. The assembler evaluates the expression to determine the shift count. All symbols in the expression must have been previously defined. The shift count must be a positive integer not exceeding 64. The assembler stores 64 minus the shift count in the jk field of the instruction. If the shift count is zero, an 054 instruction is generated. The shift is end-off with zero fill.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
055775		S7	S7>3	
055656		S6	S6>VAL+2	
054300		S3	S3>0	

- 056 Si Si,Sj<Ak Shift Si Sj left (Ak) places to Si

This instruction left shifts the 128-bit quantity formed by concatenating Si and Sj by the amount specified in Ak. The shift is end-off with zero fill. The high order 64 bits of the result are transmitted to Si. Si is cleared if the shift count exceeds 127. The shift is a left circular shift of Si if the shift count does not exceed 127 and the i and j designators are equal and non-zero. The instruction produces the same result as the 054 instruction if the shift count does not exceed 127 and the j designator is zero. The 128-bit quantity is shifted left one place if the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
056235		S2	S2,S3<A5	
056604		S6	S6,S0<A4	
056604		S6	S6<A4	(special form)
056774		S7	S7,S7<A4	

- 057 Si Sj,Si>Ak Shift Sj Si right (Ak) places to Si

This instruction right shifts the 128-bit quantity formed by concatenating Sj and Si by the amount specified in Ak. The shift is end-off with zero fill. The high order 64 bits of the result are transmitted to Si. Si is cleared if the shift count exceeds 127. The shift is a right circular shift of Si if the shift count does not exceed 127 and the i and j designators are equal and non-zero. The instruction produces the same result as the 055 instruction if the shift count does not exceed 127 and the j designator is zero. The 128-bit quantity is shifted right one place if the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
057235		S2	S3,S2>A5	
057604		S6	S0,S6>A4	
057604		S6	S6>A4	(special form)
057774		S7	S7,S7>A4	

- 060 Si Sj+Sk Integer sum of Sj and Sk to Si

This instruction forms the integer sum of Sj and Sk and enters the result into Si. No overflow is detected. Sk is transmitted to Si if the j designator is zero and the k designator is non-zero. The high order bit of Si is set and all other bits of Si are cleared if the j and k designators are both zero. The j and k designators will normally be non-zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
060237		S2	S3+S7	
060405		S4	S0+S5	

- 061 Si Sj-Sk Integer difference of Sj and Sk to Si

This instruction forms the integer difference of Sj and Sk and enters the result into Si. No overflow is detected. The negative of Sk is transmitted to Si if the j designator is zero and the k designator is non-zero. The assembler allows a special form for this case, as in the example below. The high order bit of Si is set and all other bits of Si are cleared when the j and k designators are both zero. The k designator will normally be non-zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
061123		S1	S2-S3	
061506		S5	-S6	(special form)

- 062 Si Sj+FSk Floating sum of Sj and Sk to Si

This instruction forms the sum of the floating point quantities in Sj and Sk and enters the normalized result into Si. The result will be normalized even if the operands are unnormalized. The floating point quantity in Sk is transmitted to Si as a normalized floating point number if the j designator is zero and the k designator is non-zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
062345		S3	S4+FS5	

- 063 Si Sj-FSk Floating difference of Sj and Sk to Si

This instruction forms the difference of the floating point quantities in Sj and Sk and enters the normalized result into Si. The result will be normalized even if the operands are unnormalized. The negative of the floating point quantity in Sk is transmitted to Si as a normalized floating point number if the j designator is zero and the k designator is non-zero. The k designator is normally non-zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
063761		S7	S6-FS1	

- 064 Si Sj*FSk Floating product of Sj and Sk to Si

This instruction forms the product of the floating point quantities in Sj and Sk and enters the result into Si. The result may not be normalized if the operands are unnormalized.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
064234		S2	S3*FS4	

- 065 Si Sj*HSk Half-precision rounded floating product of Sj and Sk to Si

This 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 24 bits of the result are cleared.

Example:

Code Generated

065167

location	result	operand	comments
1	10	20	35
	S1	S6*HS7	

- 066 Si Sj*RSk Rounded floating product of Sj and Sk to Si

This instruction forms the rounded product of the floating point quantities in Sj and Sk and enters the result into Si. The result may not be normalized if the operands are unnormalized.

Example:

Code Generated

066147

location	result	operand	comments
1	10	20	35
	S1	S4*RS7	

- 067 Si Sj*ISk Reciprocal iteration

This instruction forms 2 minus the product of the floating point quantities in Sj and Sk and enters the result into Si. This instruction occurs in the divide sequence as illustrated in the example for the 070 instruction.

Example:

Code Generated

067323

location	result	operand	comments
1	10	20	35
	S3	S2*IS3	

- 070 Si /HSj Floating reciprocal approximation of Sj to Si

This instruction forms an approximation to the reciprocal of the 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 shown in the example below.

Examples:

Code Generated

location	result	operand	comments
1	10	20	35
*	Divide S1	by S2; result	to S1
070320	S3	/HS2	
064113	S1	S1*FS3	
067223	S2	S2*IS3	
064112	S1	S1*FS2	

- 071 Si Ak Transmit Ak to Si with no sign extension
Si +Ak Transmit Ak to Si with sign extension
Si +FAk Transmit Ak to Si as unnormalized floating point number
Si 0.6 Transmit constant .75*2**48 to Si
Si 0.4 Transmit constant .5 to Si
Si 1. Transmit constant 1. to Si
Si 2. Transmit constant 2. to Si
Si 4. Transmit constant 4. to Si

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

When the j designator is 0, the 24-bit value in Ak is transmitted to Si. The value is treated as an unsigned integer and the high order bits of Si are cleared. When the k designator is zero, 1 is transmitted to Si.

When the j designator is 1, the 24-bit value in Ak is transmitted to Si. The value is treated as a signed integer and the sign bit of Ak is extended to the high order bits of Si. When the k designator is zero, 1 is transmitted to 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. When the k designator is zero, an unnormalized floating point 1 is transmitted to Si.

When the j designator is 3, the constant .75*2**48 is entered into Si. This constant is normally used to extract the integer part of a floating point quantity ("fix"), as illustrated in the example below.

When the j designator is 4, 5, 6, or 7, the normalized floating point constants .5, 1., 2. and 4., respectively, are transmitted to Si.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
071705		S7	A5	
071213		S2	+A3	
071122		S1	+FA2	
071130		S1	0.6	
071240		S2	0.4	
071350		S3	1.	
071460		S4	2.	
071570		S5	4.	
	*	"fix" a floating point number in S1		
	*	separate integer and fractional parts		
071230		S2	0.6	
062312		S3	S1+FS2	
023130		A1	S3	integer part
063332		S3	S3-FS2	f.p. integer part
063113		S1	S1-FS3	fractional part

- 072 Si RT Transmit RTC to Si

This instruction enters the 64-bit value of the real time clock into Si. The clock is incremented by one each clock period. The operating system clears the real time clock when the system is initialized.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
072700		S7	RT	

- 073 Si VM Transmit vector mask to Si

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

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
073200		S2	VM	

- 074 Si Tjk Transmit Tjk to Si

This instruction enters the contents of Tjk into Si. A symbolic T register number must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
074306		S3	T6	
074566		S5	T66	
074541		S5	T.TEMP	

- 075 Tjk Si Transmit Si to Tjk

This instruction enters the contents of Si into Tjk. A symbolic T register number must be previously defined.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
075306		T6	S3	
075566		T66	S5	
075541		T.TEMP	S5	

- 076 Si Vj,Ak Transmit Vj element (Ak) to Si

This instruction enters the element of Vj specified by the contents of Ak into Si. The low order 6 bits of Ak are used to determine the vector element. The second element of Vj is selected if the k designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
076456		S4	V5,A6	

- 077 Vi,Ak Sj Transmit Sj to Vi element (Ak)

This instruction enters Sj into the element of Vi specified by the contents of Ak. The low order 6 bits of Ak are used to determine the vector element. The second element of Vi receives the contents of Sj if the k designator is zero.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
077167		V1,A7	S6	

- 10h Ai exp,Ah Read from (Ah)+jkm to Ai

These instructions read words from memory directly into the A registers. The low order 24 bits of the 64-bit word are entered into Ai. The contents of Ah is added to the signed integer in the jkm field of the instruction to determine the memory address if the h designator is non-zero. Only the jkm field is used as the address if the h designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
100100004520		A1	CON1,A0	
100200004520		A2	CON1,0	
101300004521		A3	CON1+1,A1	
102417777777		A4	-1,A2	
103500000001		A5	1,A3	
104600004647		A6	CON,A4	
105700004647		A7	CON,A5	
106100000001		A1	1,A6	
107200000177		A2	177,A7	

- 11h exp,Ah Ai Store Ai to (Ah)+jkm

These instructions store words from the A registers directly into memory. The high order bits in memory are cleared. The contents of Ah is added to the signed integer in the jkm field of the instruction to determine the memory address if the h designator is non-zero. Only the jkm field is used as the address if the h designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
110100004520		CON1,A0	A1	
110200004520		CON1,0	A2	
111300004521		CON1+1,A1	A3	
112417777777		-1,A2	A4	
113500000001		1,A3	A5	
114600004647		CON,A4	A6	
115700004647		CON,A5	A7	
116100000001		1,A6	A1	
117200000177		177,A7	A2	

- 12h Si exp,Ah Read from (Ah)+jkm to Si

These instructions read words from memory directly into the S registers. The contents of Ah is added to the signed integer in the jkm field of the instruction to determine the memory address if the h designator is non-zero. Only the jkm field is used as the address if the h designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
120100004520		S1	CON1,A0	
120200004520		S2	CON1,0	
121300004521		S3	CON1+1,A1	
122417777777		S4	-1,A2	
123500000001		S5	1,A3	
124600004647		S6	CON,A4	
125700004647		S7	CON,A5	
126100000001		S1	1,A6	
127200000177		S2	177,A7	

- 13h exp,Ah Si Store Si to (Ah)+jkm

These instructions store words from the S registers directly into memory. The contents of Ah is added to the signed integer in the jkm field of the instruction to determine the memory address if the h designator is non-zero. Only the jkm field is used as the address if the h designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
130100004520		CON1,A0	S1	
130200004520		CON1,0	S2	
131300004521		CON1+1,A1	S3	
132417777777		-1,A2	S4	
133500000001		1,A3	S5	
134600004647		CON,A4	S6	
135700004647		CON,A5	S7	
136100000001		1,A6	S1	
137200000177		177,A7	S2	

- 140 Vi Sj&Vk Logical product of Sj and Vk to Vi

This instruction forms the logical products (AND) of elements of Vk with Sj and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set to 1 when the corresponding bits of Sj and the element of Vk are 1 as in the following example:

```
                (Sj) = 1100
element of Vk  = 1010
element of Vi  = 1000
```

The elements of Vi are cleared if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

```
140123
```

location	result	operand	comments
1	10	20	35
	V1	S2&V3	

- 141 Vi Vj&Vk Logical product of Vj and Vk to Vi

This instruction forms the logical products (AND) of elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set to 1 when the corresponding bits of the elements of Vj and Vk are 1 as in the following example:

```
element of  Vj  = 1100
element of  Vk  = 1010
element of  Vi  = 1000
```

The i designator must not equal the j or k designator. Elements of Vj are transmitted to Vi when the j and k designators are equal.

Example:

Code Generated

```
141257
141033
```

location	result	operand	comments
1	10	20	35
	V2	V5&V7	
	V0	V3&V3	

- 142 Vi Sj!Vk Logical sum of Sj and Vk to Vi

This instruction forms the logical sums (inclusive OR) of elements of Vk with Sj and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set to 1 when one of the corresponding bits of Sj and the element of Vk is 1 as in the following example:

```
          (Sj) = 1100
element of Vk  = 1010
element of Vi  = 1110
```

Elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Examples:

Code Generated

142615

location	result	operand	comments
1	10	20	35
	V6	S1!V5	

- 143 Vi Vi!Vk Logical sum of Vj and Vk to Vi

This instruction forms the logical sums (inclusive OR) of elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set to 1 when one of the corresponding bits of the elements of Vj and Vk are 1 as in the following example:

```
element of Vj  = 1100
element of Vk  = 1010
element of Vi  = 1110
```

The i designator must not equal the j or k designator. Elements of Vj are transmitted to Vi when the j and k designators are equal.

Example:

Code Generated

143714

location	result	operand	comments
1	10	20	35
	V7	V1!V4	

- 144 Vi Sj⊻Vk Logical difference of Sj and Vk to Vi

This instruction forms the logical differences (exclusive OR) of elements of Vk and Sj and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element are set to 1 when the corresponding bits of Sj and the element of Vk are different as in the following example:

```
          (Sj) = 1100
element of Vk  = 1010
element of Vi  = 0110
```

Elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

144267

location	result	operand	comments
1	10	20	35
	V2	S6⊻V7	

- 145 Vi Vj⊻Vk Logical difference of Vj and Vk to Vi

This instruction forms the logical differences (exclusive OR) of elements Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Bits of an element of Vi are set when the corresponding bits of the elements of Vj and Vk are different as in the following example:

```
element of Vj  = 1100
element of Vk  = 1010
element of Vi  = 0110
```

The i designator must not equal the j or k designator. Elements of Vi are cleared when the j and k designators are equal.

Example:

Code Generated

145513

location	result	operand	comments
1	10	20	35
	V5	V1⊻V3	

- 146 Vi Sj!Vk&VM Vector merge

This instruction transmits Sj or elements of Vk to Vi depending on the contents of the vector mask (VM) register. The number of elements invloved is determined by the contents of the VL register. Bit n of VM determines whether Sj or element n of Vk is transmitted to element n of Vi. Sj is transmitted if bit n is one and element n of Vk is transmitted if bit n is zero. An element of Vi is cleared when the corresponding bit of VM is one and the j designator is zero. The i and k designators cannot be equal.

Example. Assume the following initial register conditions:

```
(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
```

After executing the instruction V7 S2!V6&VM the first four elements of V7 have been modified as follows:

```
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.

Example:

Code Generated

```
146314
```

location	result	operand	comments
1	10	20	35
	V3	S1!V4&VM	

- 147 Vi Vj!Vk&VM Vector merge

This instruction transmits elements of Vj or Vk to Vi depending on the contents of the vector mask (VM) register. The number of elements involved is determined by the contents of the VL register. Bit n of VM determines whether element n of Vj or Vk is transmitted to element n of Vi. Element n of Vj is transmitted if bit n is one and element n of Vk is transmitted if bit n is zero. The i designator must not equal the j or k designator.

Example. Assume the following initial register conditions:

```
(VL) = 4
(VM) = 0 60000 0000 0000 0000 0000
element 0 of V2  = 1
element 1 of V2  = 2
element 2 of V2  = 3
element 3 of V2  = 4
element 0 of V3  = -1
element 1 of V3  = -2
element 2 of V3  = -3
element 3 of V3  = -4
```

After executing the instruction V1 V2!V3&VM the first four elements of V1 have been modified as follows:

```
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.

Example:

Code Generated

```
147567
```

location	result	operand	comments
1	10	20	35
	V5	V6!V7&VM	

- 150 Vi Vj<Ak Shift Vj left (Ak) places to Vi

This instruction shifts elements of Vj left by the amount specified in Ak and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. For each element the shift is end-off with zero fill. Elements of Vi are cleared if the shift count exceeds 63. The i and j designators cannot be equal. Elements are shifted left one place if the k designator is zero.

Example:

Code Generated

```
150123
```

location	result	operand	comments
1	10	20	35
	V1	V2<A3	

- 151 Vi Vj>Ak Shift Vj right (Ak) places to Vi

This instruction shifts elements of Vj right by the amount specified in Ak and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. For each element the shift is end-off with zero fill. Elements of Vi are cleared if the shift count exceeds 63. The i and j designators cannot be equal. Elements are shifted right one place if the k designator is zero.

Example:

Code Generated

```
151341
```

location	result	operand	comments
1	10	20	35
	V3	V4>A1	

- 152 Vi Vj,Vj<Ak Double shift Vj left (Ak) places to Vi

This instruction left shifts 128-bit quantities from Vj by the amount specified in Ak and enters the result into Vi. Element n of Vj is concatenated with element (n+1) and the 128-bit quantity is shifted left by the amount specified in Ak. The shift is end-off with zero fill. The high order 64 bits are transmitted to element n of Vi. The number of elements involved is determined by the contents of the VL register. The last element of Vj, as determined by VL, is concatenated with 64 bits of zeroes. The i and j designators cannot be equal. The 128-bit quantities are shifted left one place if the k designator is zero.

Example. Assume the following initial register conditions:

```
                (VL) = 4
                (A1) = 3
element 0 of V4  = 7
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
```

After executing the instruction V5 V4,V4<A1 the first four elements of V5 have been modified as follows:

```
element 0 of V5  = 73
element 1 of V5  = 54
element 2 of V5  = 67
element 3 of V5  = 70
```

The remaining elements of V5 are unaltered.

Example:

Code Generated

152562

location	result	operand	comments
1	10	20	35
	V5	V6,V6<A2	

- 153 Vi Vj,Vj>Ak Double shift Vj right (Ak) places to Vi

This instruction right shifts 128-bit quantities from Vj by the amount specified in Ak and enters the result into Vi. Element (n-1) of Vj is concatenated with element n and the 128-bit quantity is shifted right by the amount specified in Ak. The shift is end-off with zero fill. The low order 64 bits are transmitted to element n of Vi. The number of elements involved is determined by the contents of the VL register. 64 bits of zeroes are concatenated with the first element of Vj. The i and j designators cannot be equal. The 128-bit quantities are shifted right one place if the k designator is zero.

Example. Assume the following initial register conditions:

```
                (VL) = 4
                (A6) = 3
element 0 of V2  = 17
element 1 of V2  = 0 60000 0000 0000 0000 0005
element 2 of V2  = 1 00000 0000 0000 0000 0006
element 3 of V2  = 1 60000 0000 0000 0000 0007
```

After executing the instruction V0 V2,V2>A6 the first four elements of V0 have been modified as follows:

```
element 0 of V0  = 1
element 1 of V0  = 1 66000 0000 0000 0000 0000
element 2 of V0  = 1 30000 0000 0000 0000 0000
element 3 of V0  = 1 56000 0000 0000 0000 0000
```

The remaining elements of V0 are unaltered.

Example:

Code Generated

```
153714
```

location	result	operand	comments
1	10	20	35
	V7	V1,V1>A4	

- 154 Vi Sj+Vk Integer sum of Sj and Vk to Vi

This instruction forms the integer sums of Sj and elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. No overflow is detected. Elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

154213

location	result	operand	comments
1	10	20	35
	V2	S1+V3	

- 155 Vi Vj+Vk Integer sum of Vj and Vk to Vi

This instruction forms the integer sums of elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. No overflow is detected. The i designator must not equal the j or k designator.

Example:

Code Generated

155456

location	result	operand	comments
1	10	20	35
	V4	V5+V6	

- 156 Vi Sj-Vk Integer difference of Sj and Vk to Vi

This instruction forms the integer differences of Sj and elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. No overflow is detected. The negatives of elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

156712

location	result	operand	comments
1	10	20	35
	V7	S1-V2	

- 157 Vi Vj-Vk Integer difference of Vj and Vk to Vi

This instruction forms the integer differences of elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. No overflow is detected. The i designator must not equal the j or k designator.

Example:

Code Generated

```
157345
```

location	result	operand	comments
1	10	20	35
	V3	V4-V5	!

- 160 Vi Sj*FVk Floating product of Sj and Vk to Vi

This instruction forms the products of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. Elements of Vi are cleared if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

```
160627
```

location	result	operand	comments
1	10	20	35
	V6	S2*FV7	!

- 161 Vi Vj*FVk Floating product of Vj and Vk to Vi

This instruction forms the products of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated

```
161123
```

location	result	operand	comments
1	10	20	35
	V1	V2*FV3	!

- 162 Vi Sj*HVk Half-precision rounded floating product of Sj and Vk to Vi

This 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 result into Vi. The low order 24 bits of the result elements are cleared. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i and k designator cannot be equal.

Example:

Code Generated

162456

location	result	operand	comments
1	10	20	35
	V4	S5*HV6	

- 163 Vi Vj*HVk Half-precision rounded floating product of Vj and Vk to Vi

This instruction forms the half-precision rounded products of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The low order 24 bits of the result elements are cleared. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated

163712

location	result	operand	comments
1	10	20	35
	V7	V1*HV2	

- 164 Vi Sj*RVk Rounded floating product of Sj and Vk to Vi

This instruction forms the rounded products of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i and k designators cannot be equal.

Example:

Code Generated

164314

location	result	operand	comments
1	10	20	35
	V3	S1*RV4	

- 165 Vi Vj*RVk Rounded floating product of Vj and Vk to Vi

This instruction forms the rounded products of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
165567		V5	V6*RV7	

- 166 Vi Sj*IVk Reciprocal iteration

This instruction forms, for each element, 2 minus the product of the floating point quantity in Sj and the floating point quantity in the element of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i and k designators cannot be equal.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
166123		V1	S2*IV3	

- 167 Vi Vj*IVk Reciprocal iteration

This instruction forms, for each element pair, 2 minus the product of the floating point quantities in the elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. This instruction occurs in the divide sequence as illustrated in the example for the 174 instruction. The i designator must not equal the j or k designator.

Example:

Code Generated	location	result	operand	comments
	1	10	20	35
167456		V4	V5*IV6	

- 170 Vi Sj+FVk Floating sum of Sj and Vk to Vi

This instruction forms the sums of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. Floating point quantities in elements of Vk are transmitted to Vi as normalized floating point quantities if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

```
170712
```

location	result	operand	comments
1	10	20	35
	V7	S1+FV2	

- 171 Vi Vj+FVk Floating sum of Vj and Vk to Vi

This instruction forms the sums of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated

```
171234
```

location	result	operand	comments
1	10	20	35
	V2	V3+FV4	

- 172 Vi Sj-FVk Floating difference of Sj and Vk to Vi

This instruction forms the differences of the floating point quantity in Sj and the floating point quantities in elements of Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The negatives of floating point quantities in elements of Vk are transmitted to Vi if the j designator is zero. The i and k designators cannot be equal.

Example:

Code Generated

```
172516
```

location	result	operand	comments
1	10	20	35
	V5	S1-FV6	

- 173 Vi Vj-FVk Floating difference of Vj and Vk to Vi

This instruction forms the differences of the floating point quantities in elements of Vj and Vk and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. Underflow clears the vector element. Overflow generates an exponent of 60000. No interrupt occurs. The i designator must not equal the j or k designator.

Example:

Code Generated

location	result	operand	comments
1	10	20	35
173712	V7	V1-FV2	

- 174 Vi /HVj Floating reciprocal approximation of Vj to Vi

This instruction forms an approximation to the reciprocals of the floating point quantities in elements of Vj and enters the result into Vi. The number of elements involved is determined by the contents of the VL register. This instruction occurs in the divide sequence to compute the quotients of floating point quantities as shown in the example below.

Examples:

Code Generated

location	result	operand	comments
1	10	20	35
*	Divide elements of V1 by elements of V2;		
*	result to V6		
174320	V3	/HV2	
161413	V4	V1*FV3	
167532	V5	V3*IV2	
161645	V6	V4*FV5	
*	Divide S1 by elements of V2;		
*	result to V6		
174320	V3	/HV2	
160413	V4	S1*FV3	
167532	V5	V3*IV2	
161645	V6	V4*FV5	

- 175 VM Vj,Z Form mask on Vj as defined by k in VM
VM Vj,N
VM Vj,P
VM Vj,M

This instruction is used to create a mask in the VM register depending on the elements of Vj. Each bit of VM corresponds to an element of Vj.

If the k designator is 0, the VM bit is set when the element is zero.

If the k designator is 1, the VM bit is set when the element is non-zero.

If the k designator is 2, the VM bit is set when the element is positive. The element is considered positive when it is zero.

If the k designator is 3, the VM bit is set when the element is negative.

The number of elements tested is determined by the contents of the VL register. VM bits which correspond to untested elements of Vj are cleared.

Examples:

Code Generated

location	result	operand	comments
1	10	20	35
175050	VM	V5,Z	
175061	VM	V6,N	
175072	VM	V7,P	
175013	VM	V1,M	

- 176 Vi ,A0,Ak Read VL words to Vi from memory starting at (A0) incremented by (Ak)

This instruction is used to read 64-bit words from memory directly into elements of the V registers. The number of elements involved is determined by the contents of the VL register. A0 contains the address of the first word which is entered into the first element of Vi. Successive words are entered into consecutive elements of Vi. The signed integer in Ak is added to the address of the current word to obtain the address of the next word. 1 is added if the k designator is zero. A0 in the second operand subfield is optional. 1 may be used in the third operand subfield if the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
176201		V2	,A0,A1	
176201		V2	,,A1	
176500		V5	,,1	

- 177 ,A0,Ak Vj Store VL words from Vj to memory starting at (A0) incremented by (Ak)

This instruction is used to store the elements of V registers directly into memory. The number of elements involved is determined by the contents of the VL register. A0 contains the address in memory to receive the first element. The signed integer in Ak is added to the current address to obtain the next address. 1 is added if the k designator is zero. A0 in the second result subfield is optional. 1 may be used in the third result subfield if the k designator is zero.

Examples:

Code Generated	location	result	operand	comments
	1	10	20	35
177032		,A0,A2	V3	
177032		,,A2	V3	
177030		,,1	V3	

APPENDIX A

Summary of CPU instructions

The general form of a memory reference is *base, index, increment.* For scalar references the *increment* field does not appear. The *base* field is defined by an expression. The *index* and *increment* fields reference A registers. For a vector memory reference, the *base* field must presently be null. A block copy reference to the B or T registers is of the form *register, index, length. Register* is the character B or T. The *index* field references an A register and the *length* field is an expression. A consistent format for all types of instructions allows for future expansion to treat certain formats as macro instructions.

000	ERR	Error
0010	CA,Aj Ak	Set the channel (Aj) current address to (Ak) and begin the sequence
0011	CL,Aj Ak	Set the channel (Aj) limit address to (Ak) and terminate the sequence
0012	CI,Aj	Clear the channel (Aj) interrupt flag
0013	XA Aj	Enter the XA register with (Aj)
0014	RT Sj	Enter the real time clock register with (Sj)
002	VL Ak	Transmit Ak to VL
003	VM Sj	Transmit Sj to VM
004	EX	Exit
005	J Bjk	Branch to (Bjk)
006	J exp*	Branch to ijkm
007	R exp	Branch to ijkm; set B00 = P
010	JAZ exp	Branch to ijkm if A0 = 0
011	JAN exp	Branch to ijkm if A0 ≠ 0

* The symbol "exp" denotes an expression.

012	JAP exp	Branch to ijkm if A0 positive
013	JAM exp	Branch to ijkm if A0 negative
014	JSZ exp	Branch to ijkm if S0 = 0
015	JSN exp	Branch to ijkm if S0 ≠ 0
016	JSP exp	Branch to ijkm if S0 positive
017	JSM exp	Branch to ijkm if S0 negative
020**	Ai exp Ai #exp	Transmit jkm to Ai
021**		Transmit jkm (1's) complement to Ai
022***		Transmit jk to Ai
023	Ai Sj	Transmit Sj to Ai
024	Ai Bjk	Transmit Bjk to Ai
025	Bjk Ai	Transmit Ai to Bjk
026	Ai PSj	Population count of Sj to Ai
027	Ai ZSj	Leading zero count of Sj to Ai
030	Ai Aj+Ak	Integer sum of Aj and Ak to Ai
031	Ai Aj-Ak	Integer difference of Aj and Ak to Ai
032	Ai Aj*Ak	Integer product of Aj and Ak to Ai
033	Ai CI	Channel number of highest priority interrupt request to Ai (Aj = 0)
	Ai CA,Aj	Current address of channel (Aj) to Ai (Aj ≠ 0, k = 0)
	Ai CE,Aj	Error flag of channel (Aj) to Ai (Aj ≠ 0, k = 1)
034	B,Ai,exp ,A0	Read jk+1 words starting at B-register (Ai) from memory starting at (A0)
035	,A0 B,Ai,exp	Store jk+1 words starting at B-register (Ai) to memory starting at (A0)
036	T,Ai,exp ,A0	Read jk+1 words starting at T-register (Ai) from memory starting at (A0)

** 020 or 021 instruction generated depending on the value of exp.

*** 022 instruction generated if (1) all symbols in exp have been previously defined, (2) exp not preceded by # and (3) the value of exp is less than 64.

037	,A0 T,Ai,exp	Store jk + 1 words starting at T-register (Ai) to memory starting at (A0)
040*	Si exp	Transmit jkm to Si
	Si #exp	
041*		Transmit jkm (1's) complement to Si
042	Si <exp	Form ones mask in Si from the right
	Si #>exp	
043	Si >exp	Form ones mask in Si from the left
	Si #<exp	
044	Si Sj&Sk	Logical product of Sj and Sk to Si
045	Si #Sk&Sj	Logical product of Sj and complement of Sk to Si
046	Si Sj\Sk	Logical difference of Sj and Sk to Si
047	Si #Sj\Sk	Logical difference of Sk and Sj complement to Si
050	Si Sj!Si&Sk	Logical product of Si and complement of Sk OR'ed with the logical product of Sj and Sk to Si
051	Si Sj!Sk	Logical sum of Sj and Sk to Si
052	S0 Si<exp	Shift Si left jk places to S0
053	S0 Si>exp	Shift Si right jk negative places to S0
054	Si Si<exp	Shift Si left jk places
055	Si Si>exp	Shift Si right jk negative places
056	Si Si,Sj<Ak	Shift Si Sj left (Ak) places to Si
057	Si Sj,Si>Ak	Shift Sj Si right (Ak) places to Si
060	Si Sj+Sk	Integer sum of Sj and Sk to Si
061	Si Sj-Sk	Integer difference of Sj and Sk to Si
062	Si Sj+FSk	Floating sum of Sj and Sk to Si
063	Si Sj-FSk	Floating difference of Sj and Sk to Si

* 040 or 041 instruction generated depending on the value of exp.

064		Si Sj*FSk	Floating product of Sj and Sk to Si
065		Si Sj*HSk	Half-precision rounded floating product of Sj and Sk to Si
066		Si Sj*RSk	Full-precision rounded floating product of Sj and Sk to Si
067		Si Sj*ISk	Two minus the floating product of Sj and Sk to Si
070		Si /HSj	Floating reciprocal approximation of Sj to Si
	j		
071	0	Si Ak	Transmit Ak to Si with no sign extension
	1	Si +Ak	Transmit Ak to Si with sign extension
	2	Si +FAk	Transmit Ak to Si as unnormalized floating point number
	3	Si 0.6	Transmit constant .75*2**48 to Si
	4	Si 0.4	Transmit constant .5 to Si
	5	Si 1.	Transmit constant 1. to Si
	6	Si 2.	Transmit constant 2. to Si
	7	Si 4.	Transmit constant 4. to Si
072		Si RT	Transmit RTC to Si
073		Si VM	Transmit vector mask to Si
074		Si Tjk	Transmit Tjk to Si
075		Tjk Si	Transmit Si to Tjk
076		Si Vj,Ak	Transmit Vj element (Ak) to Si
077		Vi,Ak Sj	Transmit Sj to Vi element (Ak)
10h		Ai exp,Ah	Read from (Ah)+jkm to Ai (A0=0)
11h		exp,Ah Ai	Store Ai to (Ah)+jkm (A0=0)
12h		Si exp,Ah	Read from (Ah)+jkm to Si (A0=0)
13h		exp,Ah Si	Store Si to (Ah)+jkm (A0=0)
140		Vi Sj&Vk	Logical product of Sj and Vk to Vi
141		Vi Vj&Vk	Logical product of Vj and Vk to Vi
142		Vi Sj!Vk	Logical sum of Sj and Vk to Vi
143		Vi Vj!Vk	Logical sum of Vj and Vk to Vi
144		Vi Sj\Vk	Logical difference of Sj and Vk to Vi
145		Vi Vj\Vk	Logical difference of Vj and Vk to Vi

146	Vi Sj!Vk&VM	Transmit Sj if VM bit = 1, transmit Vk if VM bit = 0 to Vi
147	Vi Vj!Vk&VM	Transmit Vj if VM bit = 1, transmit Vk if VM bit = 0 to Vi
150	Vi Vj<Ak	Shift Vj left (Ak) places to Vi
151	Vi Vj>Ak	Shift Vj right (Ak) places to Vi
152	Vi Vj,Vj<Ak	Double shift Vj left (Ak) places to Vi
153	Vi Vj,Vj>Ak	Double shift Vj right (Ak) places to Vi
154	Vi Sj+Vk	Integer sum of Sj and Vk to Vi
155	Vi Vj+Vk	Integer sum of Vj and Vk to Vi
156	Vi Sj-Vk	Integer difference of Sj and Vk to Vi
157	Vi Vj-Vk	Integer difference of Vj and Vk to Vi
160	Vi Sj*FVk	Floating product of Sj and Vk to Vi
161	Vi Vj*FVk	Floating product of Vj and Vk to Vi
162	Vi Sj*HVk	Half-precision rounded floating product of Sj and Vk to Vi
163	Vi Vj*HVk	Half-precision rounded floating product of Vj and Vk to Vi
164	Vi Sj*RVk	Rounded floating product of Sj and Vk to Vi
165	Vi Vj*RVk	Rounded floating product of Vj and Vk to Vi
166	Vi Sj*IVk	Two minus the floating product of Sj and Vk to Vi
167	Vi Vj*IVk	Two minus the floating product of Vj and Vk to Vi
170	Vi Sj+FVk	Floating sum of Sj and Vk to Vi
171	Vi Vj+FVk	Floating sum of Vj and Vk to Vi
172	Vi Sj-FVk	Floating difference of Sj and Vk to Vi
173	Vi Vj-FVk	Floating difference of Vj and Vk to Vi
174	Vi /HVj	Floating reciprocal approximation of Vj to Vi

	k		
175	0	VM Vj,Z	VM = 1 where Vj = 0
	1	VM Vj,N	VM = 1 where Vj ≠ 0
	2	VM Vj,P	VM = 1 where Vj positive
	3	VM Vj,M	VM = 1 where Vj negative
176		Vi ,A0,Ak	Read VL words to Vi from memory starting at (A0) incremented by (Ak)
177		,A0,Ak Vj	Store VL words from Vj to memory starting at (A0) incremented by (Ak)

Special forms recognized by the assembler

Ai	Ak	Transmit Ak to Ai	
Ai	Aj+1	Transmit Aj+1 to Ai	
Ai	Aj-1	Transmit Aj-1 to Ai	
Ai	-Ak	Negative of Ak to Ai	
B,Ai,exp	,	B,Ai,exp	,A0
,	B,Ai,exp	,A0	B,Ai,exp
T,Ai,exp	,	T,Ai,exp	,A0
,	T,Ai,exp	,A0	T,Ai,exp
Si	<100	Full word of ones to Si	
Si	>100		
Si	<0	Transmit zero to Si	
Si	>0		
Si	0		
Si	1	Transmit one to Si	
Si	#Sk	Complement of Sk to Si	
Si	Sk	Transmit Sk to Si	
Si	Si<Ak	Shift Si left (Ak) places	
Si	Si>Ak	Shift Si right (Ak) places	
Si	-Sk	Negative of Sk to Si	
Ai	exp,0	Ai	exp,A0
exp,0	Ai	exp,A0	Ai
Si	exp,0	Si	exp,A0
exp,0	Si	exp,A0	Si
Vi	,,Ak	Vi	,A0,Ak
Vi	,,1	Vi	,A0,A0
,,Ak	Vi	,A0,Ak	Vi
,,1	Vi	,A0,A0	Vi
S0	Si>0	S0	Si<0
S0	Si<100	S0	Si>100
Si	Si>0	Si	Si<0
Si	Si<100	Si	Si>100

APPENDIX B

Instruction timing

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 registers must be free.
4. The result register group input path must be free at execution time - 1. 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.

Execution times in clock periods are given below. An asterisk indicates that issue may be delayed because of a functional unit reservation by a vector instruction. Memory may be considered a functional unit for timing considerations.

(A=A register, M=memory, B=B register, S=S register, I=Immediate, C=Channel)

24-bit results:

A := M	10^{*}_{*}	A := C	5
M := A	1^{*}	A := A+A	2
A := B	1	A := A*A	6
B := A	1	A := pop(S)	3
A := S	1	A := lzc(S)	4
A := I	1	VL := A	1

64-bit results:

S := M	10*	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	1
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	1

Vector instructions

Four conditions must be satisfied for issue of a vector instruction:

1. The functional unit must be free.
2. The result register must be free.
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+2 clock periods except for two special cases:
   - Memory is reserved for VL+4 clock periods.
   - A shared functional unit is reserved for VL+4 clock periods if a subsequent scalar instruction requires the unit.
2. The result register is reserved for the functional unit time +(VL+2) clock periods. The result register is reserved for the functional unit time +7 clock periods if the vector length is less than 5. At functional unit time +2 (called "chain slot time") a subsequent instruction, which uses the reserved result register as an operand register and 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+1 clock periods. Vector operand registers are reserved for 6 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+5 clock periods. Scalar operand registers are not reserved.

Vector instructions produce one result per clock period. The functional unit times are given below.

functional unit	time (c.p.)
logical	2
shift	3
integer add	3
floating add	6
floating multiply	7
reciprocal approx.	14
memory	6

Memory must be quiet before issue of the B and T register block copy instructions (034-037). Subsequent instructions may not issue for 13+jk clock periods when reading data to the B and T registers (034,036). They may not issue for 5+jk clock periods when storing data (035,037).

Branch instructions may not 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 13 clock periods.

No instruction may issue in the clock period following issue of a 2-parcel instruction.

APPENDIX C

Coding examples

1. Long vectors. When vectors have more than 64 elements it is necessary to segment the vector into groups of 64 elements and a residue before processing. The following example shows an efficient way to do this.

```
            A1      FWA          vector first word address
            A2      LWA+1        vector last word address + 1
            A0      A1-A2        - vector length
            A3      A1-A2
            S2      <6
            S1      A3
            JAP     ERROR        error if vector length ≤ 0
            S1      #S1&S2
            A3      S1
            A3      A3+1         first segment length
LOOP        VL      A3           set vector length
                                 read vector segment
                                 and perform vector computations
               ...
                                 store result

            A1      A1+A3        increment current position
            A0      A1-A2
            A3      D'64
            JAN     LOOP         loop for all segments
```

2. Loop counter. An efficient way to count the number of passes through loops when the number of passes does not exceed 64.

```
            S0      <COUNT       (mask with length = loop count)
LOOP        S0      S0<1         shift mask
                                 perform computations
               ...
            JSM     LOOP         loop required number of times
```

3. Alternate tests on the contents of S registers. Usually S0 is used to test the contents of S registers for zero, non-zero, positivity or negativity. The population count and leading zero count instructions may be used to test the contents of S registers for these conditions in A0. This may be useful when S0 cannot be destroyed or when one S register test needs to be made right after another.

```
A0      PS3
JAZ     SZR         if S3 = 0

A0      PS3
JAN     SNZ         if S3 ≠ 0

A0      ZS3
JAN     SPL         if S3 ≥ 0

A0      ZS3
JAZ     SMI         if S3 < 0
```

4. Circular shifts. The double shift instructions (056 and 057) may be used to shift an S register circularly.

```
        S7      S7,S7<A2
or:     S7      S7,S7>A2
```

APPENDIX D

Use of the NOVA CAL assembler

Name: CAL

Format: CAL filename

Purpose: To assemble a CAL assembly language source file. Output may be an absolute binary file, a listing file, or both.

Switches:

Global: By default, output of an assembly is an absolute binary file (no listing file). Switches other than those specified are ignored.

/E - list only lines with errors on listing file; no effect if L or P switches not selected
/L - listing file is produced on filename.LS
/N - no absolute binary file is produced
/O - override effect of LIST pseudo-instructions; no effect if L or P switches not selected
/P - listing on printer; overridden by L switch
/X - produce cross referencing of symbol table; no effect if L or P switches not selected

Local: None

Extensions: On input, search for filename.

On output, produce filename.SV for absolute binary and filename.LS for listing (global L switch selected).

The source file name specified on the call cannot have an extension and is limited to ten characters.

Examples: CAL Z)

causes assembly of CAL source file Z, producing an absolute binary file called Z.SV.

CAL/N/L A)

causes assembly of file A, producing as output a listing file A.LS. No binary file is produced.

CAL/P/X EXAMP)

causes assembly of file EXAMP, producing an assembly listing with cross-referenced symbol table, output to the line printer, and an absolute binary file EXAMP.SV.

APPENDIX E

Assembly errors

Error Type	Definition
O	OPERAND FIELD ERROR Indicates any of a number of possible errors in the operand field. For example: - symbol or name greater than 8 characters - expression does not have proper attribute - data error; 8 or 9 encountered in octal data - syntax error
L	LOCATION FIELD ERROR Symbol in location field is erroneous.
D	DOUBLY DEFINED SYMBOL Symbol previously defined; the first definition holds.
U	UNDEFINED SYMBOL Reference to a symbol that is not defined.
R	RESULT FIELD ERROR Indicates any of a number of possible errors in the result field. For example: - symbol or name greater than 8 characters - expression does not have proper attribute - data error; 8 or 9 encountered in octal data - syntax error - ABS or ORG following instructions or = - location field symbol begins beyond column 2

APPENDIX F

Description of binary output

The absolute binary output consists of a program descriptor table (PDT) followed by a single text table (TXT) containing the absolute code.

PDT Format:

Word 0	Bits	00-03*	Table code (17)
		04-27	Word count (7)
		28-41	Number of external names (0)
		42-55	Number of entry names * 2 (2)
		56-63	Number of blocks referenced * 2 (2) (absolute block only)
Word 1	Bits	00-63	Program name (left-justified, zero fill)
Word 2	Bits	40-63	Program length
Word 3	Bits	00-63	Enter point name (left-justified, zero fill)
Word 4	Bits	00-63	Entry value
Word 5	Bits	00-63	Date (DD/MM/YY)
Word 6	Bits	00-63	Time (HH:MM:SS)

TXT Format:

Word 0	Bits	00-03	Table code (16)
		04-27	Word count (program length + 1)
		40-63	Load address

Words 1 through "program length" contain the absolute code.

*Bit positions are numbered in decimal; the high order bit is position 0.

CRAY RESEARCH, INC.

GENERAL OFFICE • P.O. Box 169, Chippewa Falls, WI 54729 • (715) 723-0266
SALES OFFICE • 7850 Metro Parkway, Suite 213, Minneapolis, MN 55420 • (612) 854-7472

CRAY RESEARCH, INC.

PRELIMINARY SITE PLANNING

CRAY–1 COMPUTER

CRAY-1 COMPUTER

Preliminary Site Planning

CRAY-1 COMPUTER

The computer consists of the mainframe (CPU), power cabinet, two condensing units, and two motor-generators and control cabinets.

Computer room environment:

60° F to 80° F (16° C to 27° C)

35% to 65% R.H.

Voltages required:

480 volt 60 Hz

115 volt 60 Hz

False floor:

12-inch (30.5 cm) minimum height (subfloor to top of false floor)

Floor to be grounded to earth ground.

Site access requirements:

Doors:

Double doors with minimum clearance of 80-inch (203 cm) height and 60-inch (152 cm) width. If ramp and door are in close relationship required door height may be greater.

Ramp:

Maximum slope of 1 in 5.

General:

Access route to site must be strong enough to support rolling casters with 1500-pound loads. That applies especially to any ramps where the unit is most vulnerable. Customer should have 1/4-inch thick Masonite (not plywood) available for covering false floor and tiled areas during the moving-in process.

CRAY-1 CPU

Description:

Refrigerant-cooled unit that contains the modules and D.C. power supplies. Unit breaks down into one piece approximately 54 inches (137 cm) in diameter by 75 inches (191 cm) high, plus six power supply sections of slightly triangular shape of roughly 38 inches (97 cm) by 24 inches (61 cm) by 18 inches (46 cm) high. Entire assembled unit weighs about 10,000 pounds (4536 kg), with the center part of the unit weighing 6000 pounds (2722 kg). Floor loading for the center section is 7 pounds to the square inch (.5 kg/cm^2).

Wiring (see Power Wiring schematic):

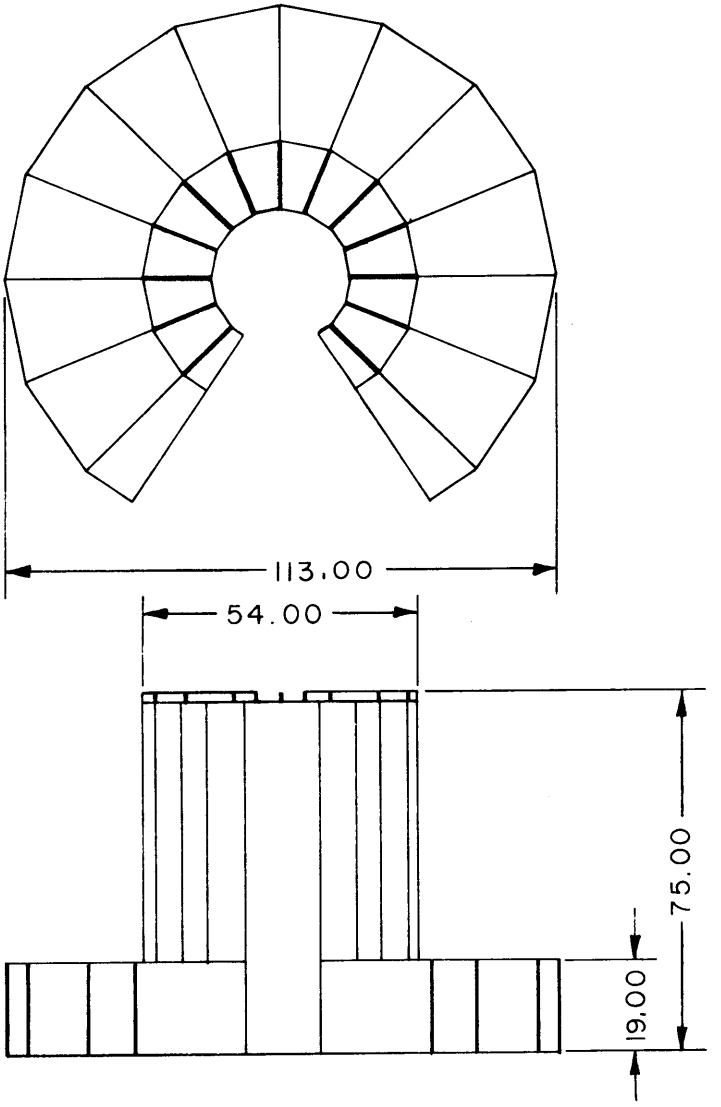
Unit requires thirty-six 3-wire circuits to the power cabinet plus two optional 115-volt circuits for convenience outlets. All additional cables are furnished by Cray Research, Inc.

Cooling (see Underfloor Refrigeration drawing):

Refrigerant piping must be installed, tested, and ready for hookup prior to the computer delivery. CRI will attach hoses between the prepared lines and the computer during the installation.

Floor preparation (see Floor Cutout drawing):

Unit requires accurate placement and sizing of floor cutouts. All floor stringers so noted must be removed and additional floor supports added where specified.



CRAY I COMPUTER

CPU

POWER REQ'D: TWO, 15 AMP, 115 V, 60 Hz CIRCUITS
(MAIN COMPUTER POWER FROM POWER CABINET)

APPROX. WT.: 10,000 lbs

CRAY-1 POWER CABINET

Description:

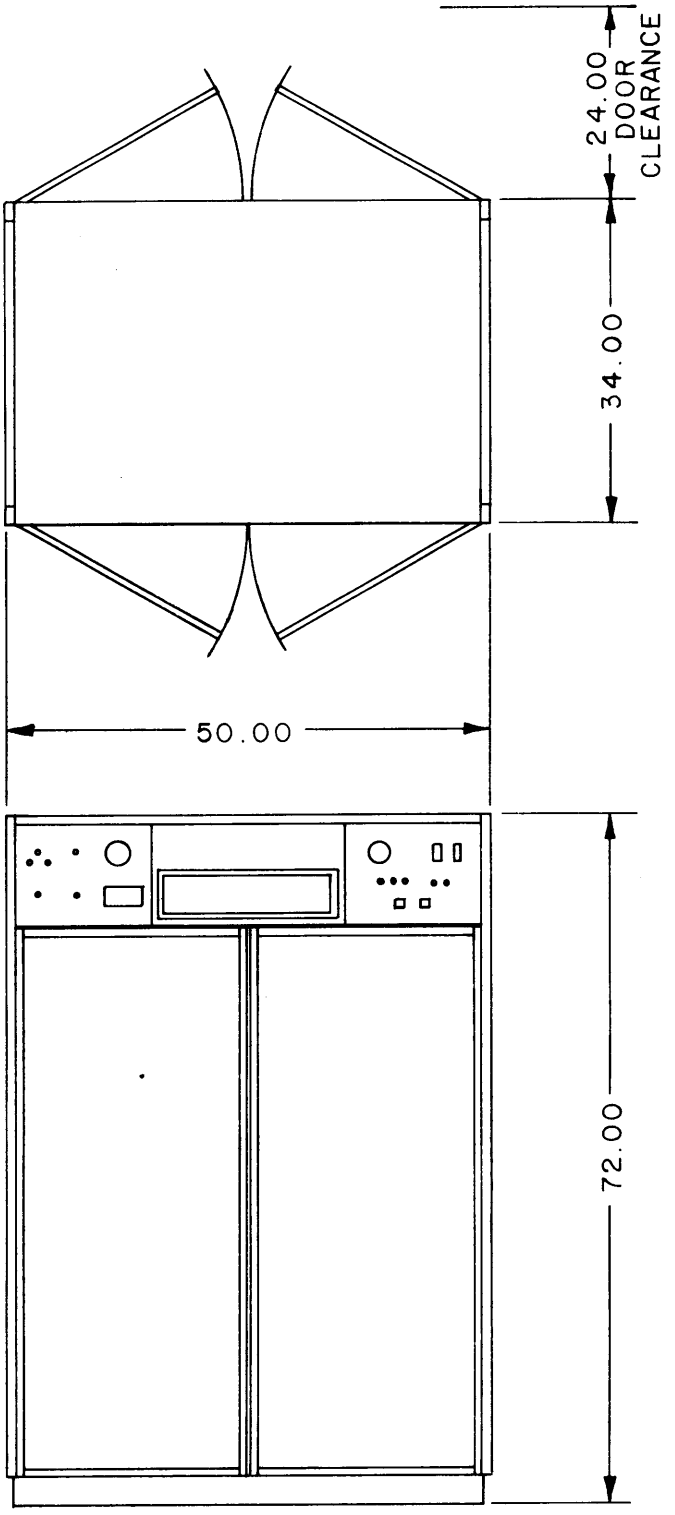
Fan-cooled (room air) unit that contains the power supplies variable transformers, system ON-OFF control, warning circuits, temperature monitoring, plus other related functions.

Wiring (see Power Wiring schematic):

Customer to install 400-Hz and 60-Hz circuits, motor-generator and condensing unit control wires, plus the 36 circuits to the CPU power supplies. CRI will install miscellaneous signal cables.

Floor preparation (see Floor Cutout drawing):

Cut out floor tile and remove stringer as shown. Additional floor support may be required to keep cut floor tile in place.



CRAY I COMPUTER

POWER DISTRIBUTION CABINET

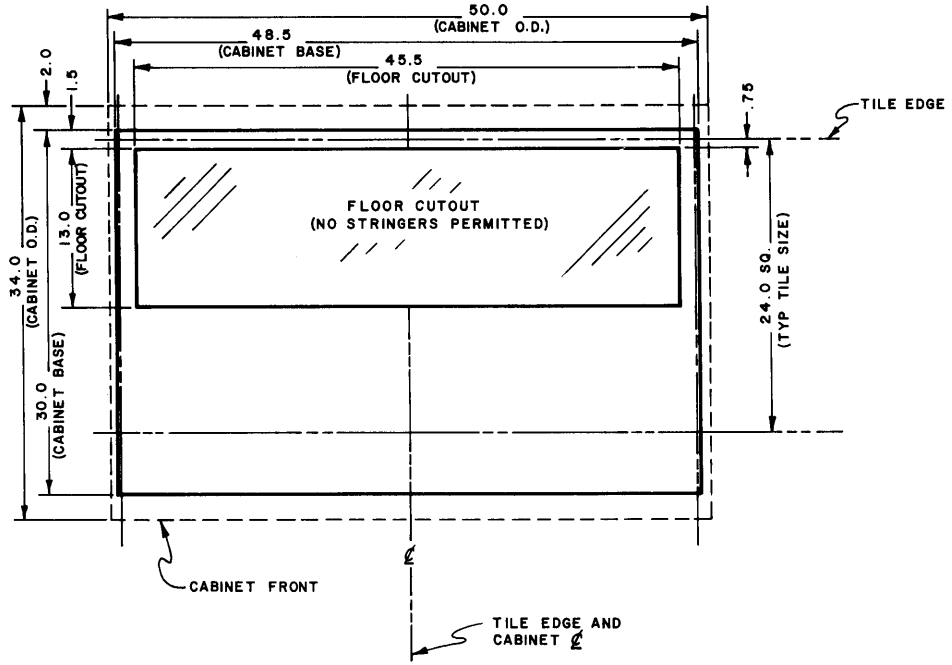
POWER REQ'D: SIX, 60 AMP, 208 V, 3ø, 400 Hz CIRCUITS

ONE, 15 AMP, 115 V, 60 Hz CIRCUIT

APPROX. WT.: 1500 lbs

APPROX. ROOM HEAT LOAD: 4000 BTU/HR

REVISIONS					
REV.	ECO.	DESCRIPTION	DRFT.	DATE	APP'D



NOTE:

1. FRONT 3 INCHES OF CUTOUT USED FOR ENTRANCE OF 6-60 AMP, 3 PHASE, 400 Hz CIRCUITS(1 1/2" KO'S)
2. REAR 8 INCHES OF CUTOUT USED FOR POWER CABLES TO MAIN FRAME(36-3/4" KO'S) PLUS 8-1/2" KO'S FOR MISC. CABLES.

PART NUMBER	REV.

ITEM	REQ'D	PART NUMBER	DESCRIPTION	MATERIAL

LIST OF MATERIAL

MATERIAL

FINISH

REMOVE ALL BURRS
BREAK SHARP EDGES & CORNERS

DO NOT SCALE

CRAY RESEARCH
INCORPORATED

PROJECT

	BY	DATE
DRWN	[illegible]	6/6/75
CHK'D		
APP.		

TOLERANCES (EXCEPT AS NOTED)
FRAC. —
DEC. —
ANG. —

TITLE
POWER CABINET
INSTALLATION INFORMATION
CRAY-1 COMPUTER

PREFIX	SIZE	PART NUMBER	REV.
	C		

SCALE 1/8 WEIGHT SHEET OF

CRAY-1 CONDENSING UNIT
(two required)

Description:

Twenty-horsepower water-cooled condenser with controls for high, low, and oil pressures. Cabinet can be furnished with skins, but if units are installed in equipment room side skins will not be furnished.

Wiring (see Power Wiring schematic and Condensing Unit Installation drawing):

Customer installed.

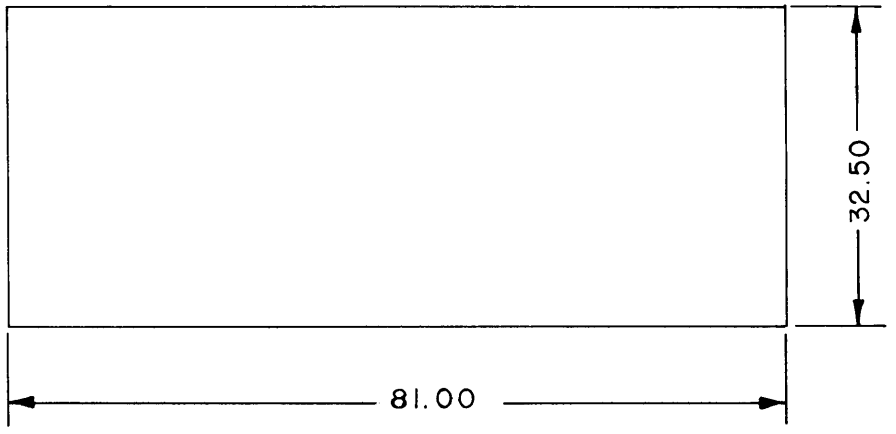
Piping (see Condensing Unit Installation drawing):

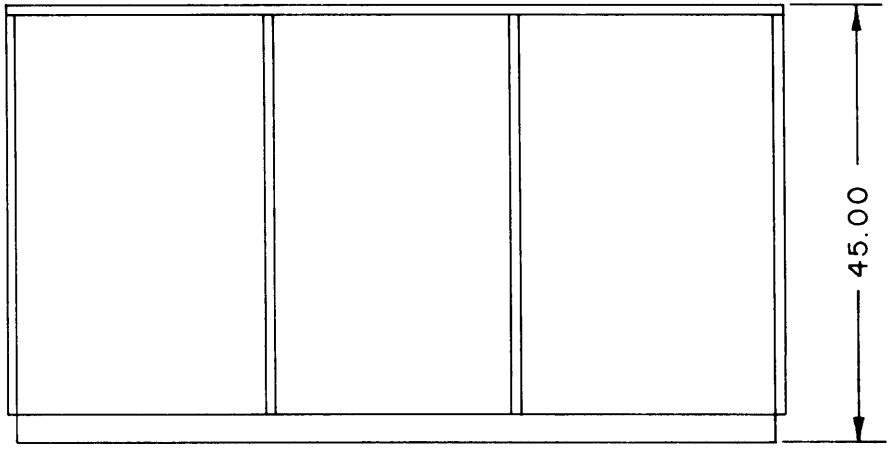
Water:

1-1/4-inch pipe, locations per drawing.

Refrigerant:

Sizes per drawing. Customer to provide access fittings for line evacuation after unit is attached. Specific drawings for individual site will be furnished by CRI.





CRAY I COMPUTER

CONDENSING UNIT

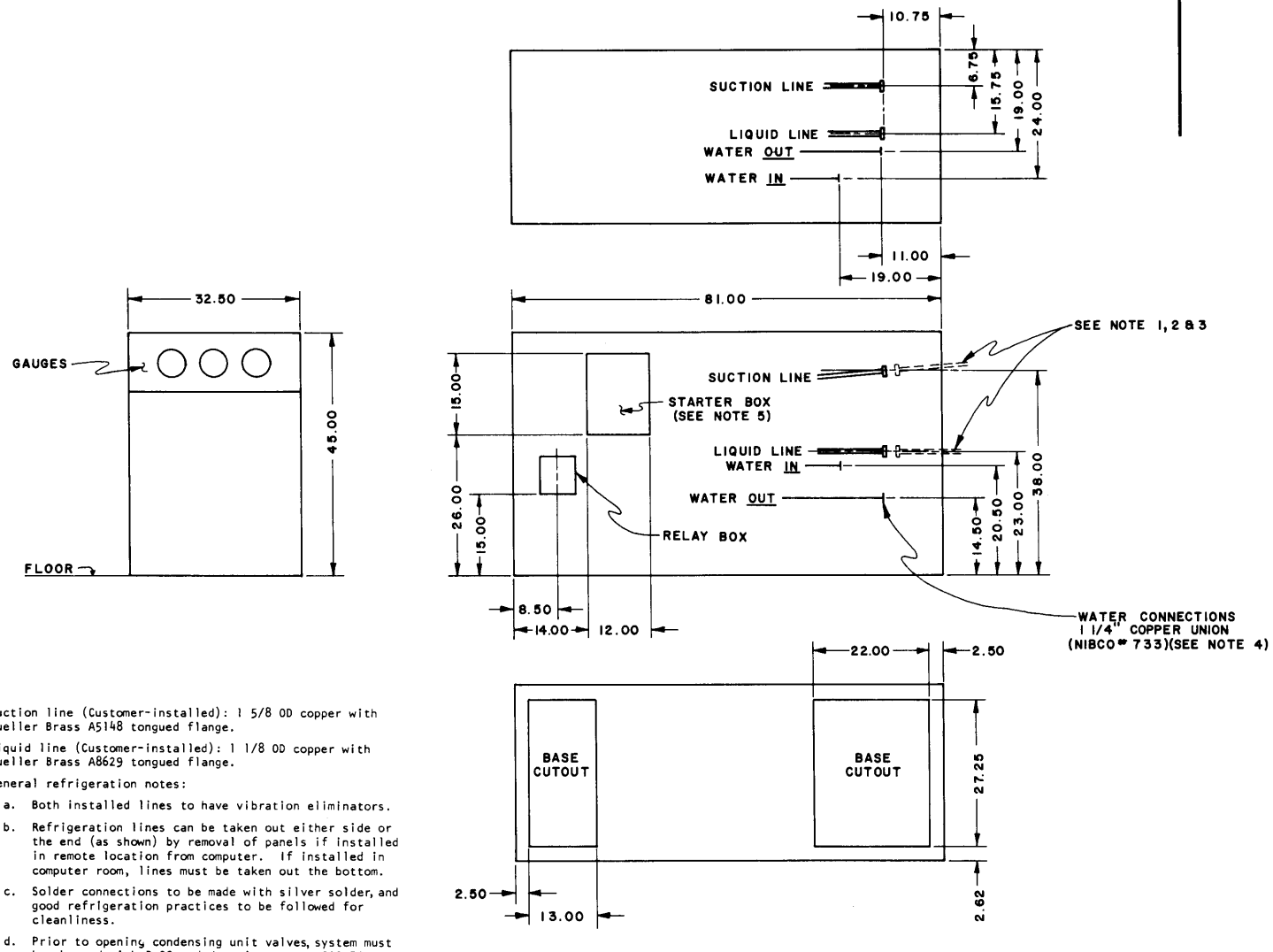
POWER REQ'D: 50 AMP, 460 V, 3ø, 60 Hz CIRCUIT

WATER REQ'D: 45° TO 75° F (20 GPM, 3 PSI DROP at 75° F)

APPROX. WT.: 1400 lbs

ESTIMATED ROOM HEAT LOAD: 18000 BTU/HR

REV.	ECO.	DESCRIPTION	DRFT.	DATE	APP'D



PART NUMBER	REV.

NOTE:

1. Suction line (Customer-installed): 1 5/8 OD copper with Mueller Brass A5148 tongued flange.
2. Liquid line (Customer-installed): 1 1/8 OD copper with Mueller Brass A8629 tongued flange.
3. General refrigeration notes:
   a. Both installed lines to have vibration eliminators.
   b. Refrigeration lines can be taken out either side or the end (as shown) by removal of panels if installed in remote location from computer. If installed in computer room, lines must be taken out the bottom.
   c. Solder connections to be made with silver solder, and good refrigeration practices to be followed for cleanliness.
   d. Prior to opening condensing unit valves, system must be charged with R-22 and dry nitrogen to 300 PSIG and tested with an electronic detector that detects leaks at 1/2 oz/year or less.
   e. Refer to general "under-floor" piping arrangement. Specific installation information will be furnished by CRI to fit the site and the computer to be shipped.
4. Water required: 45° to 75° (20 GPM, 3 PSI drop @ 75° F). If available water is over 75° F, factory must be advised. Water can be brought out similar to refrigeration lines (see note 3b). The union half (which is part of Customer's plumbing) will be furnished by CRI.
5. Power (50 amp, 460 volt, 3 phase, 60 Hz) for unit is brought into the bottom, top, or left side of the starter box. If computer room installed, bottom only.
6. Relay box connections made through the bottom only with 2 cond, 14 gauge wire to second condensing unit or power cabinet.

ITEM	REQ'D	PART NUMBER	DESCRIPTION	MATERIAL

LIST OF MATERIAL

MATERIAL

FINISH

REMOVE ALL BURRS
BREAK SHARP EDGES & CORNERS

DO NOT SCALE

CRAY RESEARCH INCORPORATED

PROJECT

	BY	DATE
DRWN	[illegible]	6/9/75
CHK'D		
APP.		

TOLERANCES (EXCEPT AS NOTED)
FRAC. — ±
DEC. — ±
ANG. — ±

TITLE
CONDENSING UNIT
INSTALLATION INFORMATION
CRAY-1 COMPUTER

PREFIX	SIZE	PART NUMBER	REV.
	C		

SCALE	WEIGHT	SHEET
1/16		OF

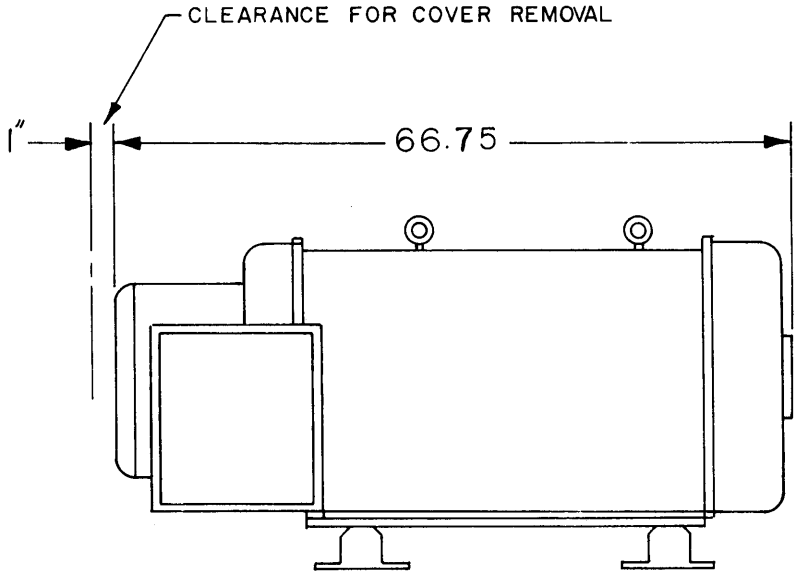
CRAY-1 MOTOR-GENERATOR
(two required)

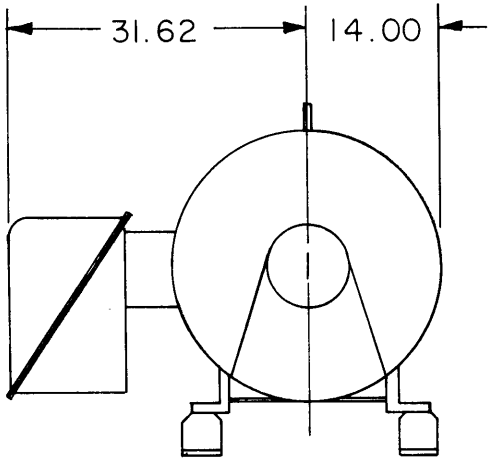
Description:

150 KVA, 208 volt, 3 phase, 400 Hz output motor-generator. 200 HP, 480 volt, 3 phase, 60 Hz drive motor and a 10 HP pony start motor. Each motor-generator with a control cabinet.

Wiring:

Units are wired both for running with the outputs hooked together through the control cabinets for alternate operation and for standby usage.





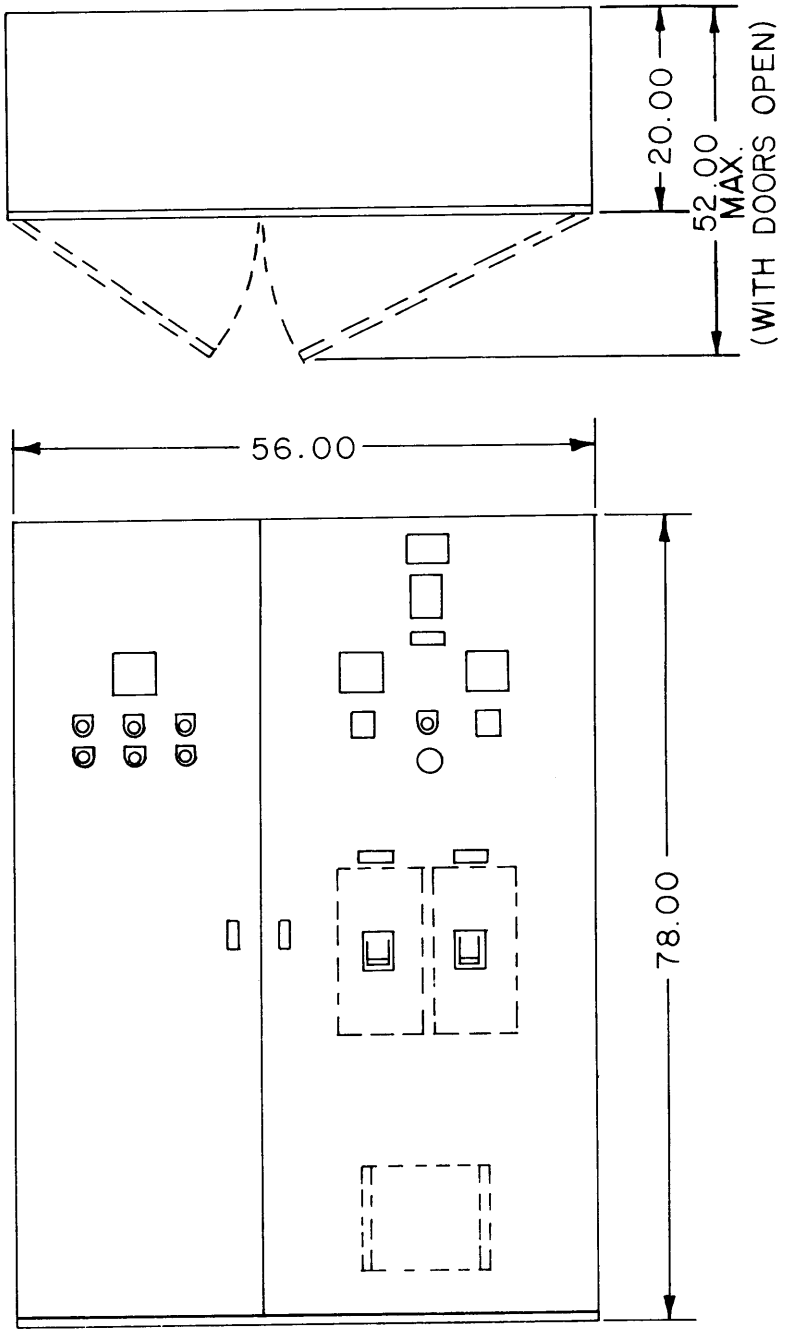
150 KVA MOTOR GENERATOR

250 AMP, 460 V, 3ø, 60 Hz

200 HP MOTOR, 10 HP PONY-MOTOR

APPROX. WT.: 3900 lbs

APPROX. ROOM HEAT LOAD: 90,000 BTU/HR



CONTROL CABINET, MOTOR GENERATOR

APPROX. WT.: 1100 lbs

CRAY-1 MCU I/O SYSTEM

This system consists of a CPU, tape drive, and controls in a rack-mounted package plus a disk pack drive, line printer, card reader, and two terminal displays. A four-foot desk and a stand are provided for the terminals.

CPU-tape drive unit:

Fan cooled using room air. Attached drop cord for 208-volt power and external male receptacle for standard 115-volt extension cord.

Disk-pack drive:

Power from control in above unit. Power and logic cables furnished.

Line printer:

Standard 115-volt extension cord--furnished by CRI.

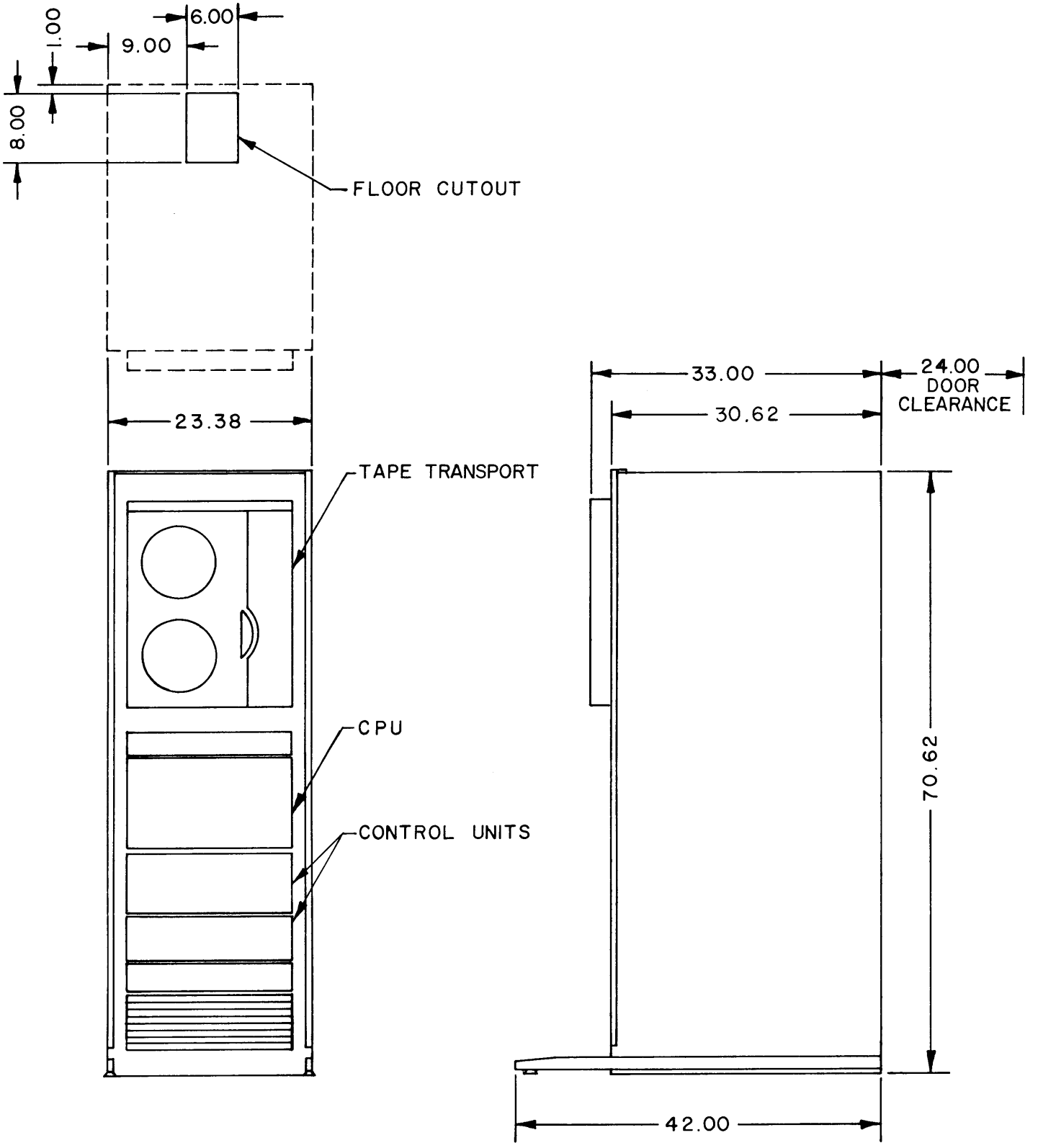
Card reader:

Standard 115-volt extension cord--furnished by CRI.

Terminal displays:

Standard 115-volt extension cord--furnished by CRI.

Customer is to provide the means to plug in all of the above equipment.



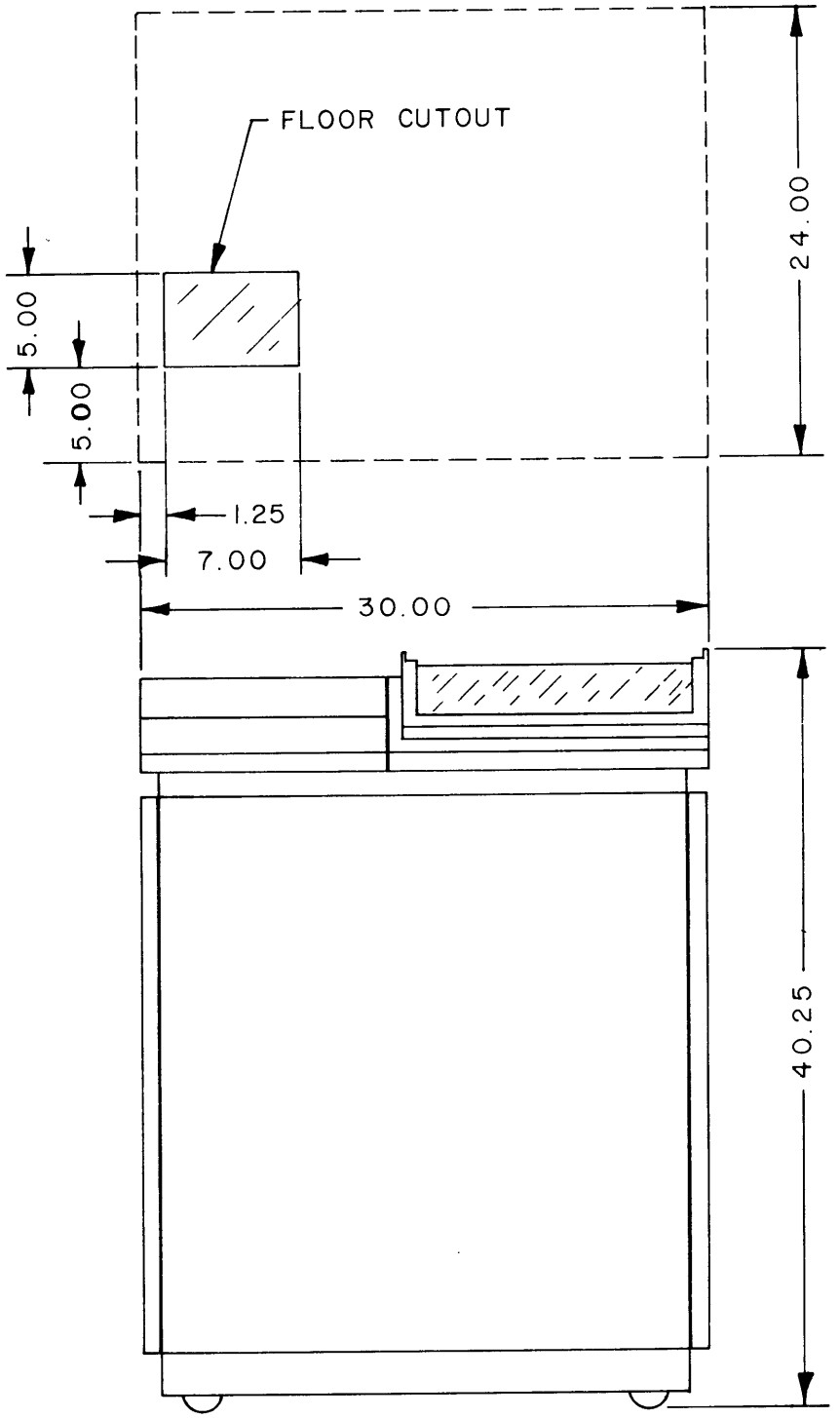
CPU, TAPE & CONTROL UNIT

CRAY I MCU

POWER REQ'D: 208 VAC, 3Ø, 60 Hz, 20 AMP CIRCUIT
115 VAC, 60 Hz, 15 AMP CIRCUIT

WEIGHT: APPROX: 500 lbs

ROOM HEAT LOAD: APPROX. 6000 BTU/HR



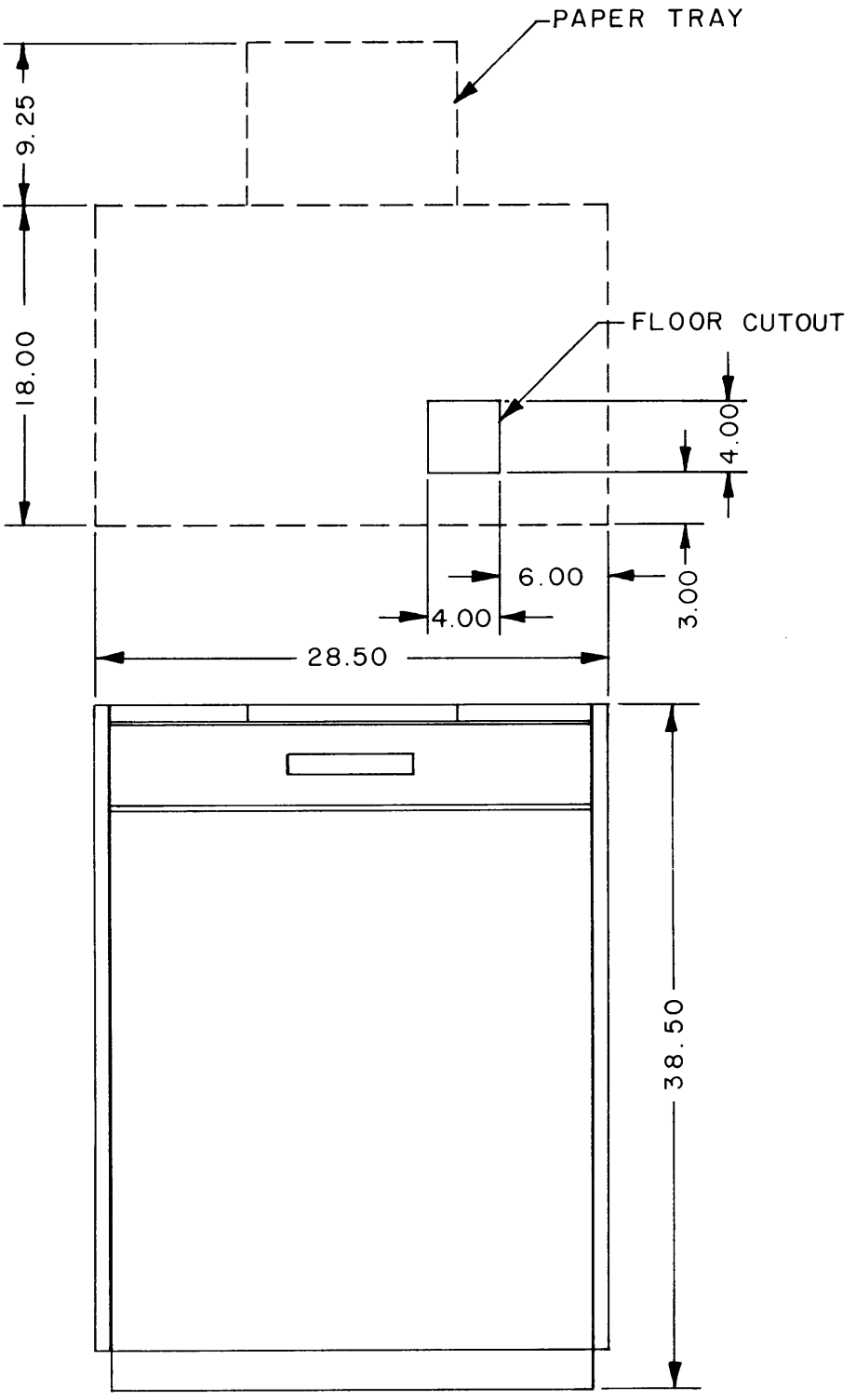
DISC-PACK DRIVE

CRAY I MCU

POWER REQ'D: FROM CONTROL UNIT - 208 OR 230 VAC, SINGLE PHASE, 60 Hz

APPROX. WT.: 350 lbs

ROOM HEAT LOAD: 1570 BTU/HR



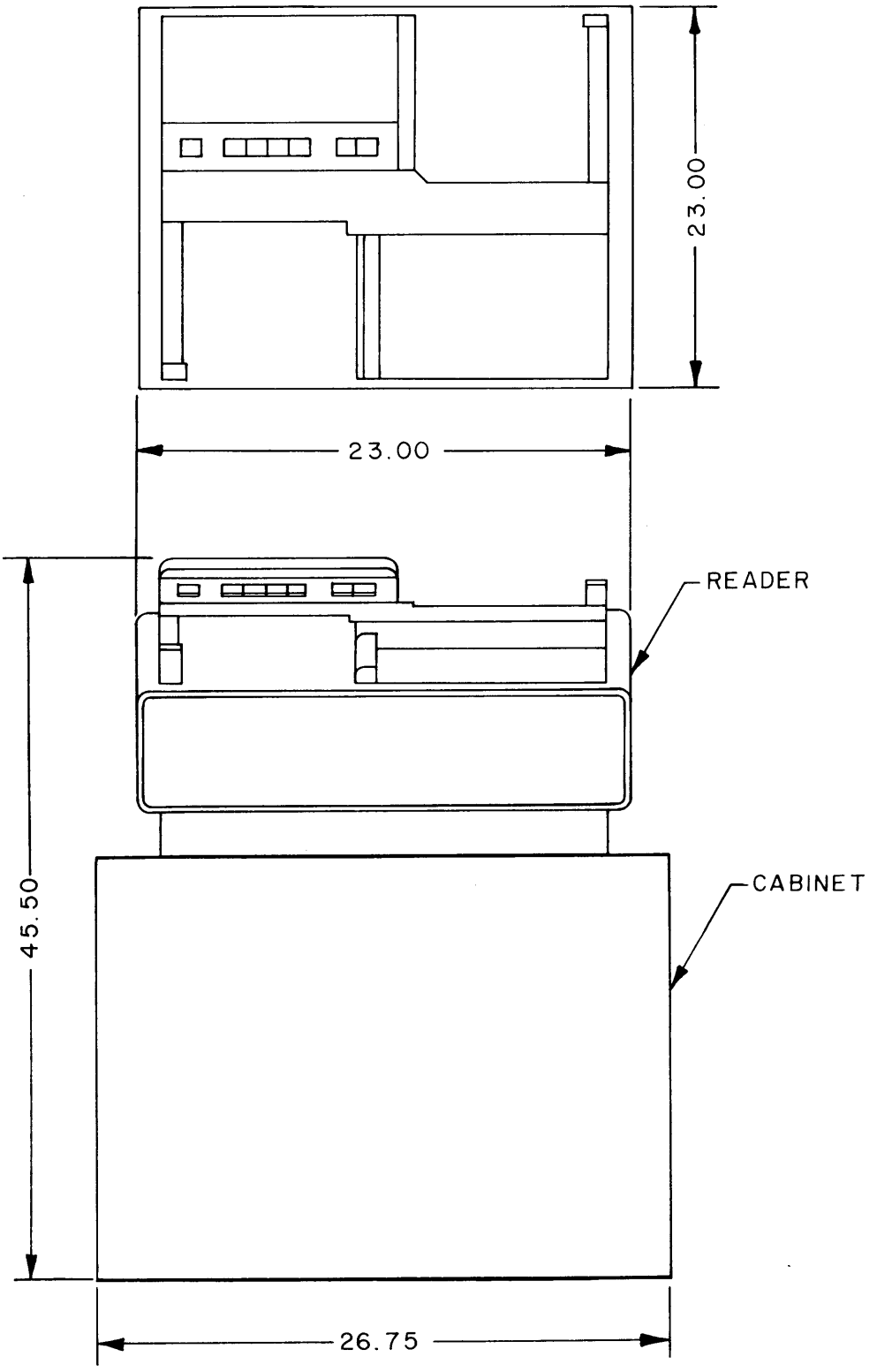
PRINTER

CRAY I MCU

POWER REQ'D: 115 VAC, 1 Ø, 60 Hz, 15 A CIRCUIT

WEIGHT: 195 lbs

ROOM HEAT LOAD: 2000 BTU/HR



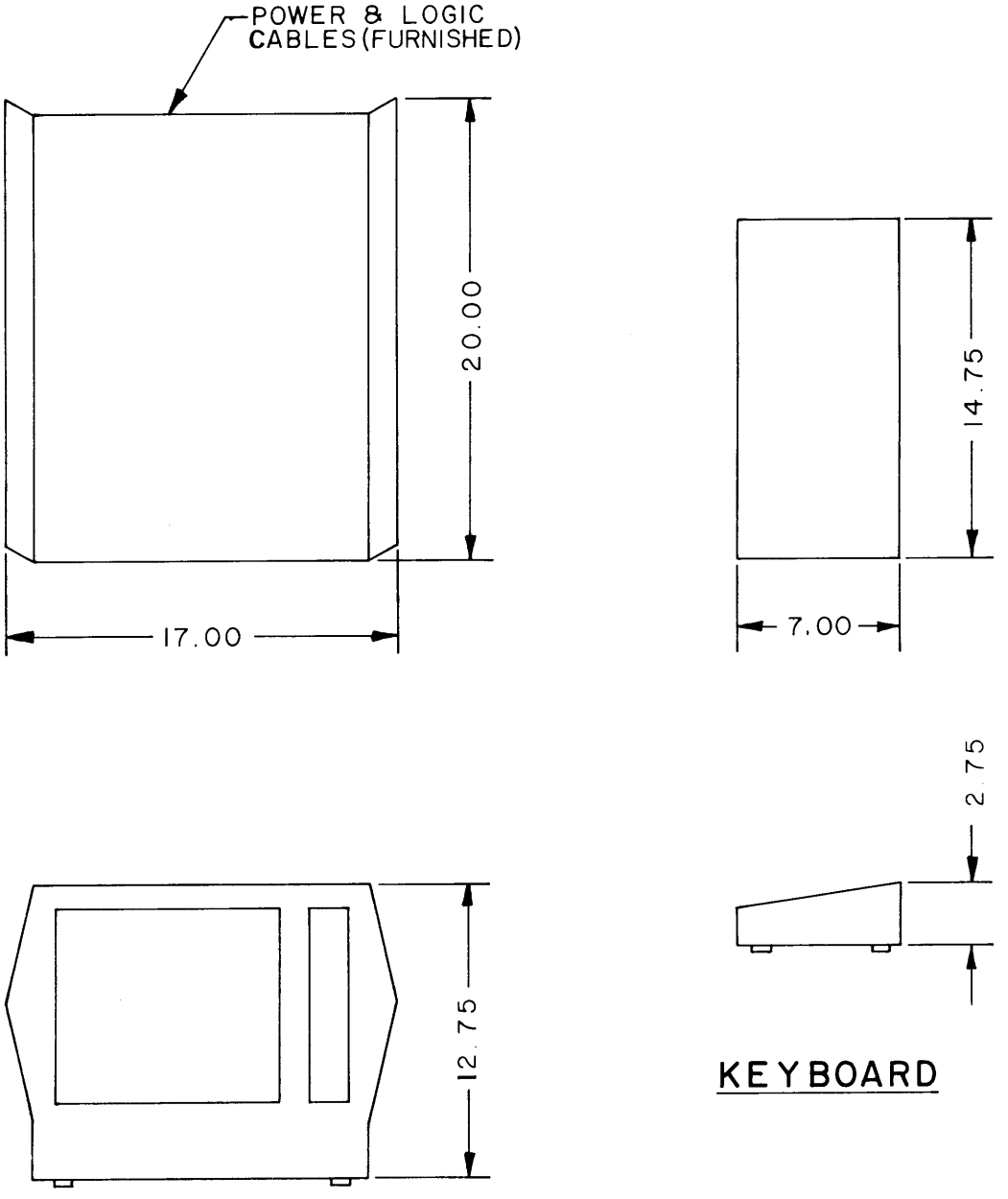
CARD READER

CRAY I MCU

POWER REQ'D: 115 VAC, 1Ø, 60 Hz, 15 AMP CIRCUIT

WEIGHT: 160 lbs

ROOM HEAT LOAD: 2050 BTU/HR(MAX)



DISPLAY

KEYBOARD

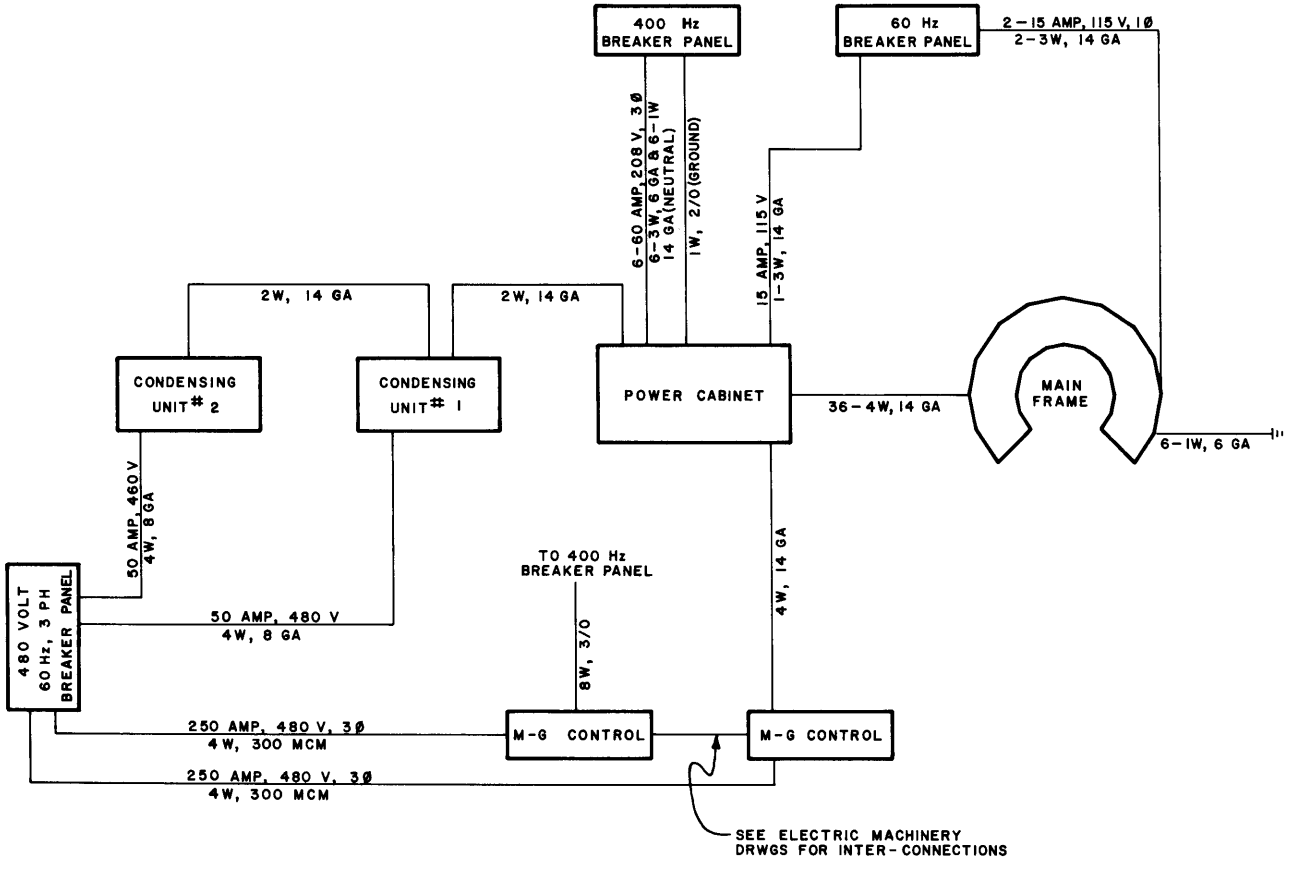
DISPLAY TERMINAL

CRAY I MCU

POWER REQ'D: 115 V, 60 Hz, 15 A CIRCUIT

HEAT DISSIPATION: 350 BTU/HR

REVISIONS					
REV.	ECO.	DESCRIPTION	DRFT.	DATE	APP'D



PART NUMBER	REV.

NOTE:

1. WIRE SIZES SHOWN MAY BE CHANGED DEPENDING UPON LOCAL CODES, WIRE INSULATION USED AND LENGTH OF RUNS.

ITEM	REQ'D	PART NUMBER	DESCRIPTION	MATERIAL

LIST OF MATERIAL

MATERIAL

FINISH

REMOVE ALL BURRS
BREAK SHARP EDGES & CORNERS

DO NOT SCALE

CRAY RESEARCH INCORPORATED

PROJECT CRAY-1

	BY	DATE
DRWN	[illegible]	6/6/75
CHK'D		
APP.		

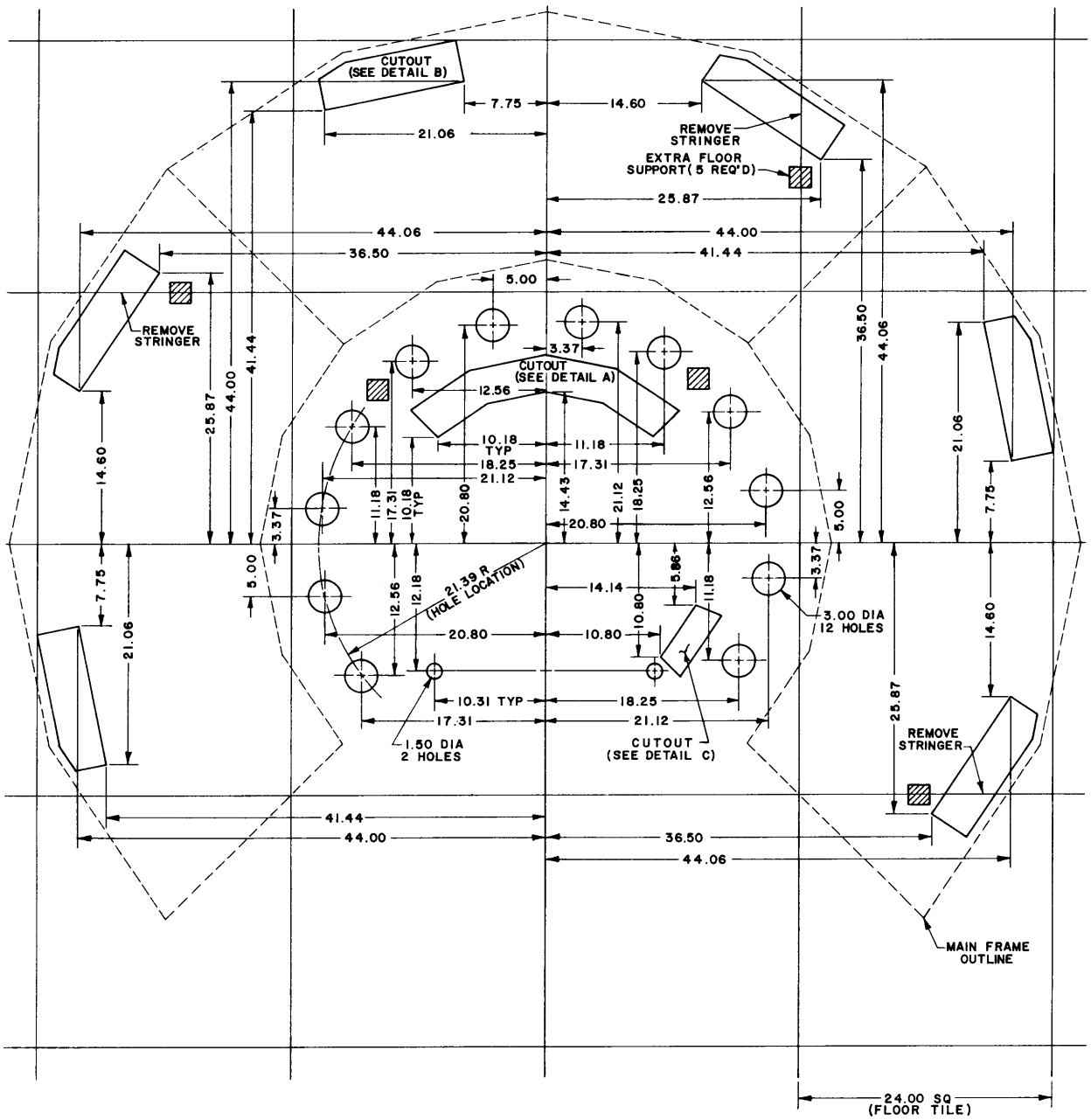
TOLERANCES (EXCEPT AS NOTED)
FRAC. — ±
DEC. — ±
ANG. — ±

TITLE
POWER WIRE SCHEMATIC (COMPUTER INSTALLATION)

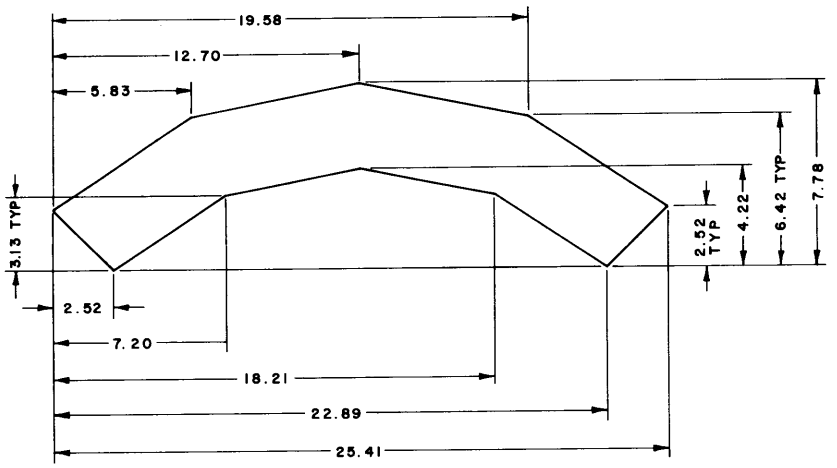
PREFIX	SIZE	PART NUMBER	REV.
	C		

SCALE | WEIGHT | SHEET OF

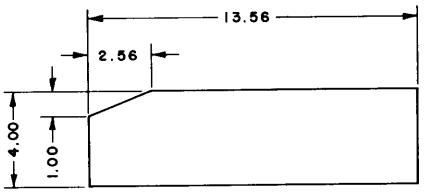
BRUNING 40-21 500-6-72


CUTOUT
(SEE DETAIL B)
7.75
14.60
21.06
REMOVE
STRINGER
EXTRA FLOOR
SUPPORT(5 REQ'D)
25.87
44.06
44.00
36.50
41.44
5.00
REMOVE
STRINGER
3.37
CUTOUT
(SEE DETAIL A)
12.56
10.18
TYP
11.18
18.25
17.31
21.12
14.43
21.12
18.25
12.56
20.80
20.80
11.18
17.31
10.18
TYP
20.80
14.60
25.87
44.00
41.44
3.37
5.00
36.50
44.06
21.06
7.75
5.00
7.75
21.06
21.39 R
(HOLE LOCATION)
12.56
12.18
14.14
5.86
11.18
10.80
3.37
3.00 DIA
12 HOLES
20.80
10.80
14.60
10.31 TYP
18.25
17.31
21.12
25.87
1.50 DIA
2 HOLES
CUTOUT
(SEE DETAIL C)
REMOVE
STRINGER
41.44
44.00
36.50
44.06
MAIN FRAME
OUTLINE
24.00 SQ
(FLOOR TILE)

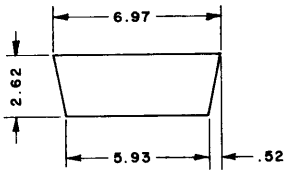
REVISIONS					
REV.	ECO.	DESCRIPTION	DRFT.	DATE	APP'D



DETAIL "A"



DETAIL "B"
(TYP 6 PLCS)



DETAIL "C"

NOTE:
1. DO NOT EDGE FLOOR CUTOUTS UNLESS EDGING IS FLUSH OR BELOW FLOOR SURFACE.

ITEM	REQ'D	PART NUMBER	DESCRIPTION	MATERIAL

LIST OF MATERIAL

PART NUMBER / REV.

MATERIAL
FINISH
REMOVE ALL BURRS
BREAK SHARP EDGES & CORNERS
DO NOT SCALE

CRAY RESEARCH INCORPORATED

PROJECT CRAY-1

	BY	DATE
DRWN	[illegible]	6/11/75
CHK'D		
APP.		

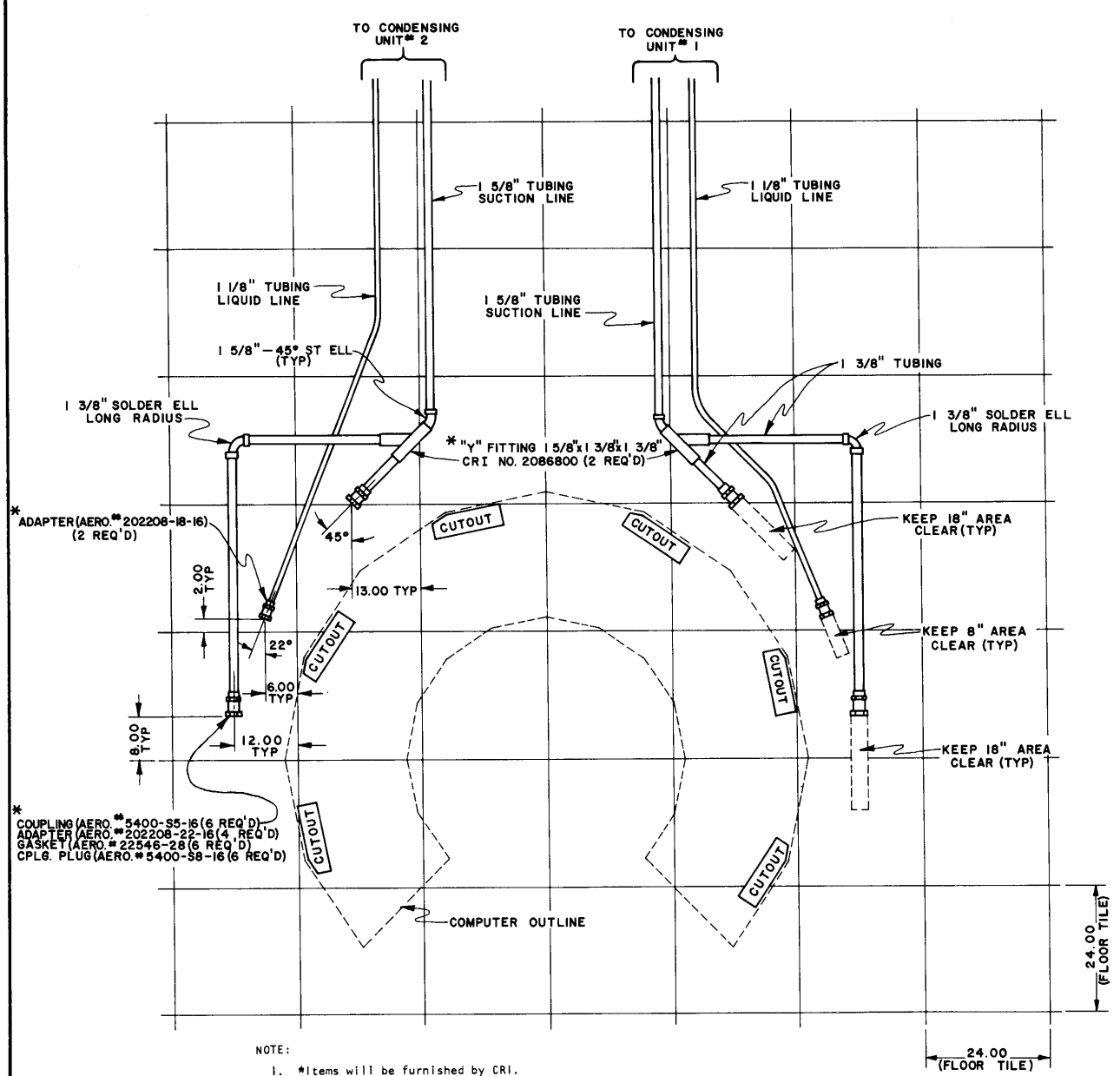
TOLERANCES (EXCEPT AS NOTED)
FRAC. — ±
DEC. — ± .12
ANG. — ±

TITLE
FLOOR CUTOUTS
MAIN FRAME
SERIAL 2 & UP

PREFIX	SIZE	PART NUMBER	REV.
	D		

SCALE 1/8 | WEIGHT | SHEET OF

REVISIONS					
REV.	ECO.	DESCRIPTION	DRFT.	DATE	APP'D



PART NUMBER	REV.

NOTE:

1. *Items will be furnished by CRI.
2. All connections to be made with silver solder.
3. Good refrigeration practices to be used--especially cleanliness.
4. System to be pumped down and hold 500 microns of vacuum.
5. System to be charged with R-22 and nitrogen to 300 PSI and leak tested with a detector with sensitivity of 1/2 oz per year or less.
6. Reevacuate system and add R-22 holding charge.
7. Piping arrangement may change--depending upon condensing unit location with respect to mainframe.
8. Specific site drawings will be furnished by CRI. These to include all refrigeration piping between mainframe and condensing units.

ITEM	REQ'D	PART NUMBER	DESCRIPTION	MATERIAL
			LIST OF MATERIAL	

MATERIAL

FINISH

REMOVE ALL BURRS
BREAK SHARP EDGES & CORNERS

DO NOT SCALE

CRAY RESEARCH INCORPORATED

PROJECT

	BY	DATE
DRWN	LWH	6/9/75
CHK'D		
APP.		

TOLERANCES (EXCEPT AS NOTED)
FRAC. — ±
DEC. — ±
ANG. — ±

TITLE
-INSTALLATION-
UNDER FLOOR REFRIG.
FOR 1/2-FULL MEMORY
COMPUTER

PREFIX	SIZE	PART NUMBER	REV.
	C		

SCALE 1/16 | WEIGHT | SHEET OF

CRAY RESEARCH, INC.

GENERAL OFFICE • P.O. Box 169, Chippewa Falls, WI 54729 • (715) 723-0266
SALES OFFICE • 7850 Metro Parkway, Suite 213, Minneapolis, MN 55420 • (612) 854-7472

CRAY 2

COMPUTING SYSTEM

December 20, 1982

Table of Contents

Introduction 1
Hardware design 2
Common memory 3
Background processing 4
Foreground processing 5

Foreground System 6
System dead start 7
Host system interface 8
DD29 disk storage unit 9
Disk system organization 10
Foreground communication channels 11
Disk controller 12
Foreground processor instruction summary 15

Background Processor 16
General description 17
Floating point data format 19
Fixed point data format 20
Instruction format 21
Vector registers 22
Scalar registers 23
Address registers 24
Local memory 25
Instruction stack 26
Instruction issue 27
Floating point multiply unit 28
Floating point add unit 31
Vector mask register 34
Vector length register 35
Real time clock 35
Semaphore flags 36
Status register 37
Channel interface 40
Instruction summary 42

Introduction

The Cray-2 computing system is an evolution of the Cray-1 system and provides an order of magnitude performance improvement at a comparable price. There are several factors which make this possible.

1) Large Semiconductor Memory

The semiconductor industry has made giant strides forward in the development of large integrated circuit memory chips in the decade since the Cray-1 computer was designed. A million words of direct access memory was the largest memory practical at the time of the original Cray-1 design. This was later expanded to four million words as larger memory circuits became available. This size is the limit that addressing allows in a Cray-1 system.

The Cray-2 system contains a direct access memory of 256 million words. This increase of almost two orders of magnitude changes the way that large scientific problems can be solved.

2) Multi-processors

The Cray-2 system contains four independent background processors each somewhat faster than a Cray-1 computer. These four processors can be brought to bear on a single large computation using the common 256 million word memory. Computation is time interlocked with high speed semaphore flags which control the step advances during the computation.

3) Integrated Foreground Processing

The Cray-2 system brings control of the peripheral support equipments into the main frame hardware. A single foreground processor coordinates the data flow into and out of the system common memory. The synchronous operation of this foreground processor with the four background processors allows an order of magnitude increase in data throughput over the original Cray-1 system.

4) Liquid Immersion Cooling

The Cray-2 system operates in a small liquid filled main frame cabinet. The circuits are cooled by direct liquid contact with the integrated circuit packages. This liquid immersion technology allows greatly reduced physical size and higher speed computation than is possible with previous techniques.

Hardware Design

The Cray-2 hardware is constructed of synchronous networks of binary circuits. These circuits are packaged in 320 pluggable modules. The modules each contain 750 integrated circuit packages. Total integrated circuit population in the system is 240000 units of which 75000 are memory.

The pluggable modules are three dimensional structures with an array of circuit packages 8 by 8 by 12 units. There are eight printed circuit boards which form the module structure. Circuit interconnections are made in all three directions within the module. External dimensions of the module are one inch by four inches by eight inches. One end of the module contains a circuit connector which mates with a connector in the cabinet frame. This connector has 288 pairs of pins for twisted pair wire communication between the modules in the cabinet frame.

Modules are arranged in the cabinet frame in 14 columns each 24 modules high. The columns are arranged in a portion of a circle with a 20 degree angle between columns. An inert electronic liquid circulates in the cabinet frame and flows through the module circuit boards across the four inch surface. Liquid velocity is one inch per second through the modules. Total module column height is 24 inches.

The semicircle of module columns is located on top of a similar structure containing power supplies for the system. Total power consumption for the system is 180 kilowatts. Total cabinet height including the power supplies is 43 inches.

There are 20 types of integrated circuit packages used in the logical networks of the machine. The circuits consist of emitter coupled logic gates with a maximum gate width of six inputs. Total gate capacity of the circuit packages is 16. Most of the 20 types of circuit packages contain two levels of gates within the package. The package has 16 connecting pins.

A 250 megahertz oscillator controls the timing throughout the circuit modules in the machine. The oscillator signal is transmitted as a square wave over 120 ohm twisted pair wires to each of the module connectors. Wire lengths are controlled so that the travel time to the individual modules is accurate within 100 picoseconds. The oscillator square wave is delivered to each individual circuit package within the module. An 800 picosecond pulse is formed from the square wave to gate data into register latches within the packages. This 800 picosecond strobe pulse occurs simultaneously throughout the machine with a period of 4 nanoseconds. This time is referred to as the machine clock period.

Common Memory

The Cray-2 system common memory consists of 128 storage banks of two million words each. Each word consists of 64 data bits and eight error correction bits. This memory is shared by the foreground processor, background processors and peripheral equipment controllers. It contains program code for the background processors as well as data for problem solution. System tables are located here for the foreground processor but foreground program code is not. Data buffers for the disk files are located directly in the background job data fields in this memory.

Each two million word memory bank occupies one Cray-2 circuit module. There is an independent data path from each bank to each of four memory access ports. A background processor and a foreground communication channel are associated with each memory port. Total memory bandwidth is 64 gigabits per second. Total memory capacity is 17 gigabits. Each background processor can read or write a word of data per clock period in a vector mode.

The integrated circuits used in the common memory contain 256 thousand bits of data. The bank of memory consists of a 8 by 8 by 9 array of these circuits in a three dimensional package. The memory bank module then contains 576 of these memory circuits and 192 logic circuits to support the memory access paths. Memory access time at the circuit level is 100 nanoseconds. Memory cycle time is 160 nanoseconds.

The Cray-2 logic circuits are significantly faster than the memory circuits. This speed discrepancy is used to minimize the number of physical data paths that are necessary to connect the 128 memory bank modules to the four memory ports. Data is transported between memory bank and access port in a three clock period long packet of data. The packet is 24 bits wide. A similar packet four clock periods long is used between access port and memory bank. In this case the memory bank address precedes the write data packet.

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

Background Processing

The four background processors in the Cray-2 system are composed of processing elements similar to those in a Cray-1 processor. The Cray-2 instruction set includes similar registers and arithmetic functions. Incompatibilities exist in order to allow the Cray-2 to address the large common memory.

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

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

Resources of a background processor are summarized below.

Eight S registers - 64 bit length - For scalar operands
Eight A registers - 32 bit length - For address and integer operands
Eight V registers - 64 elements of 64 bits - For vector segments
Instruction buffer - 512 parcels of 16 bit length - For instruction loops
Local memory - 1024 words of 64 bit length - for scalar or vector backup
Floating point functional units - For computation in 64 bit floating point mode
Integer functional units - For computation in 64 bit integer mode
Address functional units - For computation in 32 bit integer mode

Foreground Processing

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

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

The majority of foreground processor activity involves data transfer between the disk file storage units and the common semiconductor memory. The system has provision for 40 disk file units. Control for these units is organized into groups of four units each. Group control allows the high speed circuits of the Cray-2 to time share the supervision of the slower speed disk units.

All disk files may read or write data at the same time. Disk files may be addressed as individual storage units, or they may be tied in synchronous groups to provide higher transfer rates than the 35 megabit per second rate of the individual disk units. Such synchronous grouping is of arbitrary size and may extend from two units to the entire system complement. Grouping is under the control of the foreground processor which decides the mapping and order of disk sectors into a common buffer.

Resources of the foreground processor are summarized below.

Eight A registers - 32 bit length - Integer operands
Local data memory - 2048 words of 32 bit length - Temporary storage
Instruction memory - 32,768 parcels of 16 bit length
Integer functional units - For computation in 32 bit integer mode
Four communication channels - 4 gigabits per second each
Capacity of 40 DD29 disk files - 200 gigabits

****

FOREGROUND SYSTEM

****

System Dead Start

The Cray-2 system is initialized from the maintenance control console. This control console is a microprocessor with diskette storage. The microprocessor is connected to the Cray-2 cabinet over an eight bit low speed communication path. The dead start process involves loading a special diskette with the proper microprocessor program to sequence the initialization.

The process begins with a control signal from the maintenance console to preset the foreground control processor in a instruction memory load mode. The data for the foreground instruction memory is then transmitted from the diskette to the foreground processor.

The background processors and common memory are forced into an idle mode during the loading of the foreground processor memory. Foreground processor execution then begins at address zero in the foreground memory. The preamble of this program code clears common memory of noise data and presets nominal conditions in the background processors.

The foreground processor then begins its normal scan of channel activity. A job may be initiated by a host system interface or by a maintenance console request for diagnostic activity.

Host System Interface

The Cray-2 system is intended to be connected into a host system. The host system would contain the data concentration and communication equipment that is appropriate for a large computational facility.

One or more wide bandwidth data paths should connect the host system equipment to the Cray-2 cabinet frame. Bandwidth is likely limited by the host system and configurations will vary depending on the type of front end equipment used.

Each data path from the host system to the Cray-2 system requires an interface module in the Cray-2 cabinet frame. Provision is made for a number of these interface modules. These modules perform the function of resynchronizing data from devices with a different clock period and buffering the data as appropriate between the two systems.

Bandwidth for each interface module is limited to one gigabit per second by the internal structure of the Cray-2 system.

An alternative to the wide bandwith data path described above is shared disk file storage. The DD29 disk files have dual access ports. One or more disk file units can be designated as shared facilities and data can move between host system and Cray-2 system through this media. Disk file data format is the same as in the Cray-1 system.

DD29 Disk Storage Unit

The Cray-2 system can support a number of DD29 disk storage units. These units are manufactured by Control Data Corporation and are currently used in Cray-1 systems.

Each DD29 disk storage unit has a capacity of 4.8 gigabits of data. This data is recorded on 40 surfaces which rotate at 3600 revolutions per minute. The rotational latency is then 16.6 milliseconds.

The data is recorded and read back from the disk surfaces by forty recording heads which can be positioned axially on the disk surface. These heads are connected in ten groups of four recording heads each. Data from the four heads in one group can be read concurrently into the disk controller.

The recording heads can be positioned on 822 tracks for each disk surface. All heads move together in the positioning process. Moving to an adjacent track requires 15 milliseconds. Moving the maximum distance requires 80 milliseconds.

Each recording track on the disk surface is divided into 18 sectors. Each sector contains 512 common memory words of data plus error detection and correction information. The position of the heads on the disk surface is verified by data recorded and read back in each sector. The data within the sector is self correcting through a Fire code polynomial generator.

Data transfer rate using the four recording heads in parallel is 35.5 megabits per second. Consecutive sectors of data may be read or written with alternating data buffers in the disk controller. The head group selection may be changed as data is moving for the last sector on a disk track. The first sector on the next track may then be processed without missing a disk revolution. The total data package which may be moved from the disk surface without positioning the recording heads is then 180 sectors. This package of data is called a disk cylinder.

Disk System Organization

The Cray-2 system can include up to 40 DD29 disk file storage units. These units can be addressed as individual storage devices as they are in a Cray-1 system. There are problems with this approach which the large common memory of the Cray-2 can avoid. These problems are the data transfer rate for individual files, the rotational latency of the disk units, and the reliability of mechanical devices.

The disk storage system on a Cray-2 has the option of operating in a synchronous mode with all disk units running in parallel in a lock step mode. For this approach to look attractive the buffer size for individual disk references must be of the order of 100,000 words. This was not practical in smaller memory machines.

Consider a system configuration with 16 DD29 disk files as an example for the synchronous mode of operation. The foreground processor is given a disk address which consists of a pseudo-track number. This is the cylinder and head group for a disk file with no flaws. A table lookup converts this pseudo-track into a physical track for each disk unit. All disk units are positioned in parallel.

The foreground processor reads angular position for each disk surface to determine the sector currently under the recording head. It then begins a data stream from common memory to disk surfaces choosing the portion of the common memory buffer appropriate for the current angular position of each disk unit. Data to 15 of the disk units is directly from the common memory job buffer. Data for the 16th disk unit is a logical difference data stream using the word by word data from the desired file. All 16 disk units write one track of data as the basic reservation unit.

On data readback the 16th disk is read concurrently with the other 15 disks. If the Fire code detectors indicate no data errors the 16th disk data is discarded. If an error has occurred it can be corrected without time loss in the data stream.

The overhead introduced by this arrangement is one disk file on fifteen. The benefit is three-fold.

1) Data rate is 525 megabits per second instead of 35 megabits per second.

2) The disk file rotational latency has gone to zero.

3) A disk file may fail completely due to head crash or motor failure with no loss of data or time.

A disk failure in this system can be corrected during system operation by removing the defective file and replacing it with another unit. The new unit can then be brought on line by running a background job which takes 2.5 minutes of disk system time to record the faulty unit data from the data on the other 15 files.

Foreground Communication Channels

There are four communication channels in the system which link the background processors, foreground processor, and peripheral equipment controllers. Most of the data traffic is between controllers and memory. Data blocks are generally 512 common memory words in length. Typical channel reservation time is ten microseconds.

Each channel has associated with it one common memory access port and one background processor control. The foreground processor is associated with all four channels in a supervisory role. Normally four peripheral controllers are associated with each channel.

Each channel consists of 16 data paths and two control paths. One control path carries a channel busy flag and the other a data ready flag. These 18 paths interconnect the members of the communication group in a continuous loop. Each member receives data on 18 bit paths each clock period and transmits that data onward to the next member in the following clock period. Data may then move about the loop from any transmitting member to any receiving member.

Channel Sequence for Function Initiation

An idle channel is indicated by a zero bit moving about the channel busy path. The data present flag and 16 data bits will also be zero values at this time. Activity is initiated for the channel by the foreground processor. The foreground processor breaks the channel loop at its node and transmits a function code on to the next member of the loop. The busy flag is set by the foreground processor and remains set as the function code moves around the loop.

The function code may be followed by a parameter for the addressed member of the communication group. The function code and parameter data are each 32 bits in length. They are encoded into four 16 bit parcels as they leave the foreground processor node.

Each member of the channel group tests for a busy flag transition from zero to one state. This transition indicates a message is passing the node from the foreground processor. Each member translates the function code to determine if action is required. If this member is addressed, and a response is required, the response is transmitted in the two clock periods following the parameter data.

The foreground processor, having initiated the function transmission, waits for the zero to one transition on its busy flag input path. It then skips the two clock periods corresponding to its own parameter data and reads the two following parcels of response data. If this completes the communication for the function, the foreground processor clears the busy flag, waits for the return transition, and the channel becomes idle.

Disk Controller

A disk controller supervises the activities of four DD29 disk files. This controller interprets function codes sent by the foreground processor over the communication channel. The controller determines its own channel call number by wiring connections in the cabinet frame.

The function code transmitted from the foreground processor on the channel link consists of 32 bits plus an optional 32 bits of parameter data. The basic 32 bit format is interpreted by the controller circuits as follows.

4 bits) Controller number
4 bits) Unit number
8 bits) Function code
16 bits) Next program address

A zero controller number is interpreted as a general call for foreground request.

Function codes are translated in the following manner.

00) Release unit
01) Reserve unit
02) Clear fault flags
03) Return to zero cylinder
04) Select margin conditions
05) Read sector number
06) Read error flags
07) Read disk status

10) Select cylinder
11) Select head group
12) Read status response
13) Enter error call address

20) Begin normal read at specified sector
21) Read correction code
22) Begin early read at specified sector
23) Begin late read at specified sector
24) Read format data in short block

30) Begin normal write at specified sector
31) Write format data

------ Function 20 - Begin normal read at specified sector

This foreground processor function call begins the process of streaming data from the disk surface to the common memory. The length of the data stream may be as short as one sector of disk data or as long as an entire disk cylinder. The steps in the data transfer are as follows.

1) Foreground to disk - Begin read at sector

The disk controller reads the sector number from the parameter word and enters it in the appropriate disk unit coincidence register. The next program address is held for call to foreground on completion.

2) Disk to foreground - Sector data ready

The disk controller senses a full sector data buffer from the sector read operation. It then responds to the next channel call with the program address from step 1.

3) Foreground to memory - Write block at address

The memory port senses this function call and prepares its input buffer for a block of data intended for the common memory address in the parameter word.

4) Foreground to disk - Transmit sector data

The foreground processor holds the channel busy flag on this call in preparation for data transmission. The disk controller sends a burst of 64 common memory data words around the loop to the memory port. A data ready flag accompanies each element. The disk controller then pauses to wait for a memory response.

5) Memory to disk - Buffer free

The memory port loads a 64 word buffer with the data from the channel. When the buffer data has been moved into common memory the memory port sends a single data ready pulse to the disk controller.

6) Disk to memory - Data burst

The disk controller receives the data ready signal and sends the next 64 word burst of data over the channel. Steps 5 and 6 repeat until the sector data transmission is complete. The disk controller then senses the last data ready pulse from the memory port and drops the channel busy flag.

Steps 2 through 6 repeat until the foreground processor requirement has been satisfied.

------ Function 30 - Begin normal write at specified sector

This foreground processor function call begins the process of streaming data from the common memory to the disk surface. The length of the data stream may be as short as one sector or as long as an entire disk cylinder. The steps in the data transfer are as follows.

1) Foreground to disk - Begin write at sector

The disk controller reads the sector number from the parameter word and enters it in the appropriate disk unit coincidence register. The program address is held for call on completion.

2) Foreground to memory - Read block from address

The memory port reads the common memory address from the parameter word and loads its 64 word buffer from that location. The channel busy flag remains set during this operation. The memory port then transmits a 64 word burst of data over the channel to the disk controller. A data ready flag accompanies each element.

3) Disk to memory - Buffer free

The disk controller loads the first 64 words of its sector data buffer with the data burst from the memory port. It then sents a data ready pulse to the memory port.

4) Memory to disk - Data burst

The memory port reads the next 64 words from the common memory and transmits the next data burst to the disk controller. Steps 3 and 4 repeat until the disk sector buffer is full. The disk controller then drops the channel busy flag which frees the channel for other use.

5) Disk to foreground - Sector data written

The disk controller moves the sector data to the write deskewing buffer for the appropriate disk unit. It begins recording on the disk surface as soon as the proper sector is under the recording head. It then responds to the next channel call with the program address from step 1.

Steps 1 through 5 repeat until the foreground processor requirements are satisfied.

Foreground Processor Instruction Summary

```
000---  Pass
001---  Pass
002i--  Enter Ai with console data
003i--  Enter console data from Ai

004i-k  Enter Ai from local memory address Ak
005i-k  Store Ai into local memory address Ak
006ijk  Enter local memory address Ai from common address Aj with length Ak
007ijk  Store local memory address Ai into common address Aj with length Ak

010ijk  Jump to Aj and hold return Ai if Ak is zero
011ijk  Jump to Aj and hold return Ai if Ak is nonzero
012ijk  Jump to Aj and hold return Ai if Ak is positive
013ijk  Jump to Aj and hold return Ai if Ak is negative

014ijk  Jump to Aj and hold return Ai if channel k is busy
015ijk  Jump to Aj and hold return Ai if channel k is idle
016ij-  Jump to Aj and hold return Ai if console data ready
017ij-  Jump to Aj and hold return Ai if console data not ready

020ijk  Enter Ai with positive jk
021ijk  Enter Ai with negative jk
022i--  Enter Ai with real time count
023---  Unassigned

024ijk  Shift Ai left jk
025ijk  Shift Ai right jk
026ijk  Enter Ai with integer sum Aj + Ak
027ijk  Enter Ai with integer difference Aj - Ak

030ijk  Enter Ai with logical product Aj and Ak
031ijk  Enter Ai with logical product Aj and complement Ak
032ijk  Enter Ai with logical difference Aj and Ak
033ijk  Enter Ai with logical sum Aj and Ak

034--k  Abort channel k
035ijk  Function Aj on channel k with response Ai.  Clear busy.
036ijk  Function Aj on channel k with parameter Ai.  Clear busy.
037ijk  Function Aj on channel k with parameter Ai.  Hold busy.

040i--  Enter Ai with positive 16 bit constant
041i--  Enter Ai with negative 16 bit constant
042i--  Enter Ai with 32 bit constant
043i--  Unassigned

044i--  Enter Ai from constant local memory address
045i--  Store Ai into constant local memory address
```

****

BACKGROUND PROCESSOR

****

General Description

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

Vector registers (V)

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

Scalar registers (S)

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

Address registers (A)

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

Local memory

There are 1024 words of local memory associated with each background processor. Each word is 64 bits in length. This memory is treated as a register file to hold scalar operands during a computation period and then return the data to the common memory.

Instruction stack

Each background processor has an instruction stack to allow program loops to execute without additional common memory references. This instruction stack contains a contiguous field of 512 parcels of program code. Programs may loop within this field using any of the branch instructions.

Floating point functions

Each background processor has dedicated functional units to perform calulations in a floating point format. There are two independent units in each processor with functions assigned in two groups as follows.

Floating point multiplication
Reciprocal approximation
Reciprocal iteration

Floating point addition
Floating point subtraction
Float to fixed point conversion

Integer functions

Each background processor has two dedicated functional units to perform fixed point calculations in a 64 bit mode. These units are similar but are specialized, one for scalar use and one for vector use. These functions may then be performed concurrently. Each unit has the following modes.

Long integer addition
Long integer subtraction
Long word data shift
Population count
Leading zero count

Logical functions

Each background processor has two dedicated functional units to perform logical operations on 64 bit data in a bit by bit mode. These units are similar but are specialized, one for scalar use and one for vector use. These functions may then be performed concurrenly. Each unit has the following modes.

Bit by bit logical product
Bit by bit logical different
Bit by bit logical sum

Address functions

Each background processor has 32 bit functional units for address or short integer computation. These units operate in a scalar mode only with the following functions.

Integer addition
Integer subtraction
Integer product

Floating Point Data Format

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

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

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

---- Coefficient

The 48 bit coefficient is a binary number with the binary point to the left of the highest order bit position. The highest order bit in the coefficent is normally a one value. The floating point number is said to be normalized if this is the case. Correct results in some functional units depend on normalized operands. The range of decimal values for the coefficient in a normalized form is 0.5 to 1.0.

---- Exponent

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

Zero:	0 00000 0000000000000000
Plus one:	0 40001 4000000000000000
Minus one:	1 40001 4000000000000000

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

Fixed Point Data Format

Fixed point computation is performed on data in a twos complement integer format. Long word arithmetic is performed with 64 bit integers. A fixed point representation of the integers zero, plus one, and minus one in this format are illustrated below using octal notation.

Zero:	0000000000000000000000
Plus one:	0000000000000000000001
Minus one :	1777777777777777777777

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

Zero:	00000000000
Plus one:	00000000001
Minus one:	37777777777

Instruction Format

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

Instruction translation involves the interpretation of four designators within the 16 bit parcel. These designators are identified below from higher to lower order bit position.

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

The f designator determines the instruction type. The f designator values are indicated in octal notation in the instruction descriptions. The i, j, and k designators generally refer to V, S, or A registers in a three address format. The i designator generally specifies the destination register for the functional computation. The j and k designators generally specify the source operands.

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

Single parcel constants are used to address the local memory. Two parcel constants are used to address the common memory. Four parcel constants are used to enter long word values in the S registers.

Vector Registers (V)

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

The instruction issue control mechanism reserves the vector registers which are involved in a functional unit streaming operation. This may be one, two, or three vector registers depending on the specific instruction. The functional unit is reserved at this same time. The instruction sequence may then proceed to the next instruction and initiate concurrent activity as long as the resources reserved are not required in the subsequent instruction.

The length of the vector stream is determined by the content of a six bit length register (L). This register is preset before the vector instruction issues and may be used for a number of vector operations of the same length. Maximum vector length is 64 elements. A zero value in the L register is interpreted as a 64 bit length indicator.

The i, j, and k designators in a vector instruction may have the same value. In the case of identical source operands the data is streamed from the same vector register to both data paths. In the case of a destination register which is the same as the source register the vector register writing function takes priority over reading but with a delay equal to the transit time of the functional unit. This may be useful as a recursive vehicle for merging vector data in some situations.

The vector registers are implemented in the hardware with 16 by 4 register chips. These integrated circuit chips have a two clock period cycle time. One clock period is required for address soak. The second clock period is then used for readout or for write strobe.

The 64 elements of a vector register are arranged in four banks of 16 elements each. Each bank has its own address register but they share a common read/write register. Consecutive element addresses for vector data move through the four banks before advancing the pointer for the first bank. This allows data to stream into, or out of, a vector register in consecutive clock periods. Data coming from common memory and moving into a vector register may come out of order with respect to the common memory access requests. This is because of quadrant and bank delays in the common memory access control mechanism.

Scalar Registers (S)

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

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

The eight scalar registers as a group have one data readout path and one data destination path. Each path may be used independently in each clock period. An instruction which requires two scalar source operands reads these operands in the two clock periods following instruction issue. The destination path must be reserved at issue time for the specific clock period of data arrival.

An exception to the destination path reservation described above is made for the common memory read to scalar register. A separate path from the common memory to the scalar registers as a group is provided because the arrival time cannot be known at issue time.

Address Registers (A)

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

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

The eight address registers as a group have one data readout path and one data destination path. Each path may be used independently in each clock period. An instruction which requires two address source operands reads these operands in the two clock periods following instruction issue. The destination path must be reserved at issue time for the specific clock period of data arrival.

Local Memory

There are 1024 words of local memory associated with each background processor. Each word is 64 bits in length. This memory is treated as a register file to hold scalar operands during a computation period and then return the data to the common memory.

The local memory is implemented in hardware with 256 by one register chips. These integrated circuit chips have a three clock period cycle time. The first clock period is used for address soak only. The second clock period is used for address soak plus chip select. The third clock period is used for data readout or for write strobe.

The local memory is organized into four banks of 256 words each. Each bank has its own address register. All four banks share a data write register and a data readout register. A scalar reference initiates all four banks at the same clock period. The proper bank is then sampled to the readout register three clock periods later for a read reference or the write strobe is enabled to the proper bank for a write reference.

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

The local memory is reserved at instruction issue time in the same manner as a functional unit. The reservation has a three clock period duration for a scalar reference. The reservation has a duration of four clock periods plus the vector length for a vector reference.

Instruction Stack

Each background processor has an instruction stack to allow program loops to execute without additional common memory references. This instruction stack contains a contiguous field of 512 parcels of program code. Programs may loop within this field using any of the branch instructions.

The instruction stack is implemented in hardware with 16 by 4 register chips. These integrated circuit chips have a two clock period cycle time. One clock period is required for address soak. The second clock period is then used for readout or for write strobe.

The instruction stack is organized into four banks of 32 words each. Each word is 64 bits in length. Each bank has its own address register. The four banks share a common data write register and a data readout register. Consecutive addresses in the instruction stack move through the four banks before advancing the pointer for the first bank. This allows the common memory data to enter the instruction stack 64 bits wide with a word each clock period.

A branch out of the instruction stack field initiates a fetch vector in the common memory. The common memory address is formed from the requested branch destination parcel address by a shift down of two bit positions. The fetch vector length is always 32 common memory words. The instruction stack is cleared and limit registers which represent the instruction stack parcel boundaries are reset to correspond with the new field. The arriving data is then entered in the first 128 parcel positions of the instruction stack.

Data is read from the instruction stack to the instruction issue control module one parcel at a time. A threshold test is made with each parcel readout to determine if the program sequence is nearing the end of the available data in the instruction stack. A new fetch vector is initiated when the readout enters the last 64 parcels of the available data.

The instruction stack limit registers are adjusted with each new fetch vector. The oldest 128 parcels of data are discarded as a full instruction stack advances.

Instruction Issue

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

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

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

Constants are intermixed with instruction parcels in the instruction stream. The constant parcels are passed through the instruction issue queue without test. An instruction with a following constant parcel may then issue and the following parcel issue without additional delay. Multiparcel constants delay the next instruction issue by one clock period for each constant parcel beyond the first.

Detail control signals are sent from the issue control module to the register modules and functional unit modules in the two clock periods following issue. These signals are formed and transmitted concurrent with the translation and issue test of the following instruction parcel. Most instructions involve a two clock period time sequence of register selection and readout for source operands.

The issue control module sends the detail control signals to the destination register module at a later time. This delay corresponds to the transit time of the source operands through the functional unit. A shift register in the issue control module provides the memory vehicle for these destination tags.

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

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

Floating Point Multiply Unit

The floating point multiply unit performs floating point calculations for both scalar and vector instructions. The unit is reserved during the period of the vector stream for the case of a vector instruction sequence. Scalar use is locked out during this period. No reservation is made at the time of issue for a scalar floating multiply instruction. Successive scalar instructions may issue at the maximum rate.

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

---- Floating Point Product

The floating point multiply unit forms the product of two operands in floating point format and delivers a result in floating point format. If both operands are normalized the result will also be normalized. Instructions which use this sequence are the following.

124ijk Enter Si with floating product Sj * Sk
154ijk Enter Vi with floating product Sj * Vk
155ijk Enter Vi with floating product Vj * Vk

The product is formed by merging the 48 bit coefficients into a matrix of logical product circuits. All combinations of coefficient bits are formed in the same clock period. The resulting 48 x 48 bits are then summed using three input adder circuits. The exponents are added to form the result exponent. A one count correction to the exponent is required if the resulting coefficient must be shifted by one bit position for normalization.

The 48 x 48 matrix of logical product bits is truncated eight bit positions below the lowest order result coefficient bit. Round bits are added to this lower field to give an equal population of high and low round errors for random operands. A round bias will exist over narrow ranges of operands because of the one bit correction shift after the round operation.

The Sj operand is held in the floating multiply unit during the execution of the 154 instruction. This frees the Sj register for other use during the vector streaming period.

Several special cases are treated in floating point multiplication for operands out of range. These cases are as follows.

1) One or both operands have overflow exponent
2) Sum of operand exponents is an overflow
3) Sum of exponents is an underflow
4) Both exponents are underflow

Cases 1 and 2 cause a floating point error signal to the background processor status register. The computational result delivered to the destination register is forced to an overflow value. Case 3 results in an all zero word sent to the destination register. Case 4 computes the coefficients with no normalize correction. Result exponent is zero for this case. This last case aids mutiple precision calculations.

---- Reciprocal Approximation

The floating point mutiply unit forms a 25 bit approximation to the reciprocal of a floating point operand value. This approximation is computed in two steps. The first step is a table lookup using a read only memory. The second step is a linear interpolation using the multiply mechanism. Instructions which use this sequence are the following

126i-k Enter Si with reciprocal approximation of Sk
157i-k Enter Vi with reciprocal approximation of Vk

The table used in the first step of the reciprocal approximation contains 4096 words of data each 26 bits long. The address for the table lookup is taken from the highest order 13 bits of the operand coefficient. The operand is assumed normalized so that only 12 bits are required in the table address.

The table readout is separated into two sections each 13 bits in length. These values are then used in a linear interpolation computation. The form of this computation is as follows.

Let A be a reciprocal approximation for the upper 13 bits of operand coefficient
Let B be the operand coefficient truncated at 26 bits
Let R be the better reciprocal approximation
Then the iteration step for interpolation is:

$$R = 2A - A*A*B$$

The two approximations read from the table are 2A and -A*A. The multiply unit is then used to form the product in the iteration formula with the additional sum included in the hardware pyramid.

There are two special cases treated in the reciprocal approximation sequence.

1) Operand exponent has overflow value
2) Operand exponent has underflow value

Both cases cause an error signal to be sent to the background processor status register. Both cases cause the computational result to be forced to an overflow value.

---- Reciprocal Iteration

The floating point multiply unit forms a floating point number which is used in a second iteration for the reciprocal of a full precision operand. The first iteration is formed in the reciprocal approximation described above. The second iteration uses the same process to form a reciprocal approximation with 47 bits of coefficient accuracy. The instructions which use this sequence are the following.

125ijk Enter Si with the reciprocal iteration 2 - Sj * Sk
156ijk Enter Vi with the reciprocal iteration 2 - Vj * Vk

The above instructions are followed by a normal floating point multiply instruction to complete the iteration.

An error signal is sent to the background processor status register if either of the operands in the above instructions has an overflow exponent. An overflow result exponent is forced in this case.

Floating Point Add Unit

The floating point add unit performs floating point calculations for both scalar and vector instructions. The unit is reserved during the period of the vector stream for the case of a vector instruction sequence. Scalar use is locked out during this period. No reservation is made at the time of issue for a scalar floating add operand. Successive scalar instructions may issue at the maximum rate.

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

---- Floating Addition

The floating point add unit forms the sum of two operands in floating point format and delivers a result in floating point format. The result will always be normalized regardless of source operand status. Instructions which use this sequence are the following.

120ijk Enter Si with the floating sum Sj + Sk
170ijk Enter Vi with the floating sum Sj + Vk
171ijk Enter Vi with the floating sum Vj + Vk

The sum is formed by first subtracting operand exponents. The sign of the exponent difference determines the larger operand. The larger operand coefficent is entered in a 49 bit adder with a one round bit in the 49th bit position. The smaller operand coefficient is entered in the adder with a zero round bit in the 49th bit position. A binary illustration of the two quantities is shown below.

```
1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1
1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx0
```

The smaller operand coefficient field is then complemented if the algebraic signs of the operands disagree. The smaller operand coefficient field is shifted down by the amount of the exponent difference. The two quantities now appear as in the binary illustrations below.

```
1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1              signs
000001xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx0            agree
-------------------------------------------------

1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1              signs
111110xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1          disagree
-------------------------------------------------
```

The two quantities are now added in the 49 bit window illustrated above to form the result coefficient. A carry from the highest order bit is retained as part of the result coefficient.

The result coefficient may be negative if the operand exponents were equal or if the operands were not normalized. The floating add unit hardware anticipates this possibility and forms a second result coefficient which includes a complement of the data and the addition of a one bit in the 48th bit position.

The next step in the floating add process involves a test of the leading zeros in the result coefficient to determine the amount of shift necessary to normalize it. An overflow of the adder means a negative correction of one bit position. Leading zeros in the result coefficient can mean as much as a 47 bit positive correction.

A special test is made for all zero bits in the result coefficient. In this case the exponent field in the result is also cleared. A word of all zeros is delivered to the destination register.

The result coefficient is shifted by the appropriate amount and a correction is made in the larger operand exponent. The algebraic sign of the resulting floating point form is computed from the types of steps in the process.

A special case test is made for one or both operands with an overflow exponent. In this case an error signal is sent to the background processor status register and an overflow exponent is forced in the result delivered to the destination register.

---- Floating Subtraction

The floating point add unit forms the difference of two operands in floating point format and delivers a result in floating point format. This mode of operation is essentially identical to the addition mode. The only difference is the toggle of the sign bit for the subtrahend operand. The instructions which use this form of the sequence are the following.

121ijk Enter Si with floating difference Sj - Sk
172ijk Enter Vi with floating difference Sj - Vk
173ijk Enter Vi with floating difference Vj - Vk

---- Floating to Fixed Conversion

The floating point add unit forms an integer representation of a floating point operand. This process is accomplished by adding the operand to a constant in floating point format and then masking the coefficient field into the result integer. The result must be complemented and a plus one added if the resulting integer is negative. This mechanism already exists in the mid sequence option of the floating point addition sequence. The instructions which use this form of the floating add sequence are the following.

132i-k Enter Si with integer form of floating Sk
174i-k Enter Vi with integer form of floating Vk

The constant which is used as the second operand for this sequence is the quantity illustrated below in octal notation.

0 40060 0000000000000000

The addition of the specified operand to this constant will shift the operand coefficient down by the desired amount to separate the integer portion. The round bit is omitted in this process. The resulting integer is at most 48 bits in length. A negative result is sign extended by special hardware to form a 64 bit integer result.

An operand with a floating point value greater than a 48 bit integer is sensed as an error condition. A signal is sent to the background processor status register and a zero result is delivered to the destination register.

Vector Mask Register (M)

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

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

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

030--k Enter mask bits where Vk has zero elements
031--k Enter mask bits where Vk has nonzero elements
032--k Enter mask bits where Vk has positive elements
033--k Enter mask bits where Vk has negative elements

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

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

146ijk Enter Vi with Sj masked into Vk
147ijk Enter Vi with Vj masked into Vk

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

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

034--k Enter M from Sk
114i-- Enter Si from M

Vector Length Register (L)

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

The L register content is interpreted modulo 64. The allowed values are one through 64. A zero value is therefore interpreted as a 64. The L register may be set either from a constant in the instruction stream or from a computed parameter. The instructions which communicate explicitly with the L register are the following.

025i-- Enter Ai from L
036-jk Enter L with jk
037--k Enter L from Ak

Real Time Clock

Each background processor has a 64 bit register which counts continuously at the clock period rate. This count value may then be used to determine the passage of real time to an accuracy of one clock period. The instruction which reads the real time clock is the following.

115i-- Enter Si with real time count

Semaphore Flags

The four background processors share access to 16 semaphore flags. These flags are single bit registers which provide time interlocks for common memory access to critical parameter fields.

The semaphore flags are continuously moving between the background processors. The flags are organized in four groups of four flags each. There are four flags in each processor in each clock period. In the following clock period the flags will have moved in a circular fashion to new positions. The reason for this rotation of flag positions is to provide each processor an exclusive access to any flag in a four clock period unit of time. The processor can then read or alter the flag without interference from another processor.

It is essential for the read and alter function to occur in the same clock period. The background processor instructions which perform these functions are the following.

004--k Jump to constant parcel if semaphore Ak clear
005--k Jump to constant parcel if semaphore Ak set
006--k Jump to constant parcel if semaphore Ak clear. Set semaphore
007--k Jump to constant parcel if semaphore Ak set. Clear semaphore

The lowest order four bits of data in A register k select the flag. The branch test and optional flag alteration occur in the same clock period.

Status Register

The background processor requires a number of mode flags and error condition flags to indicate to the foreground processor the status of the computation. These flags are collected into a background processor status register which resides in the background processor channel interface. There are 16 bits in this status register with bit significance as follows.

Enable Flags

bit 15) Unassigned
bit 14) Enable semaphore access
bit 13) Enable memory field error
bit 12) Enable floating point error

Mode Flags

bit 11) Interrupt on semaphore error
bit 10) Interrupt on memory field error
bit 09) Interrupt on floating point error
bit 08) Interrupt on error exit
bit 07) Interrupt on normal exit
bit 06) Interrupt on idle processor

Status Flags

bit 05) Semaphore error
bit 04) Memory field error
bit 03) Floating point error
bit 02) Error exit
bit 01) Normal exit
bit 00) Idle processor

A status flag is set by a transient condition in the background processor and remains set until action is taken by the foreground processor. The enable flags and mode flags are generally set and cleared by the foreground processor. An exception is the enable for floating point error and memory field error for which there is a special background processor instruction as well.

A call to the foreground processor occurs when a mode flag and the corresponding status flag are both set. The foreground processor then reads the status register to determine the reason for the call. See 'Background Processor Channel Interface' for details.

Individual bits in the status register have the following functions.

---- Bit 14 - Enable semaphore access

This bit permits the background processor program to alter the state of the interprocessor semaphore flags. If this bit is cleared an instruction execution which attempts to alter a semaphore flag is aborted and bit five in the status register is set.

---- Bit 13 - Enable memory field error

This bit enables the setting of an error status flag if the program makes a reference to a common memory address outside the prescribed field. Such a reference may be for either an operand or an instruction fetch.

---- Bit 12 - Enable floating point error

This bit enables the setting of an error status flag if a floating point calculation is made with an operand which is larger than the floating point range.

---- Bit 11 - Interrupt on semaphore error

This bit enables a foreground processor call if the semaphore error flag is also set.

---- Bit 10 - Interrupt on memory field error

This bit enables a foreground processor call if the memory field error flag is also set.

---- Bit 09 - Interrupt on floating point error

This bit enables a foreground processor call if the floating point error flag is also set.

---- Bit 08 - Interrupt on error exit

This bit enables a foreground processor call when the background processor executes an error exit instruction.

---- Bit 07 - Interrupt on normal exit

This bit enables a foreground processor call when the background processor executes a normal exit instruction.

---- Bit 06 - Interrupt on idle processor

This bit enablgs a foreground processor call when the background processor is in an idle mode.

---- Bit 05 - Semaphore error

This bit is set when enabled by a zero bit 14 in the status register and a background processor instruction attempt to alter a semaphore flag.

---- Bit 04 - Memory field error

This bit is set when enabled by bit 13 in the status register and a background processor attempt to reference common memory beyond the prescribed range.

---- Bit 03 - Floating point error

This bit is set when enabled by bit 12 in the status register and a background computation with an operand larger than the foating point range.

---- Bit 02 - Error exit

This bit is set when the background processor executes an error exit instruction.

---- Bit 01 - Normal exit

This bit is set when the background processor executes a normal exit instruction.

---- Bit 00 - Idle processor

This bit is set when the background processor is in an idle mode.

Channel Interface

The background processors have an interface to the foreground processor for the control information necessary to sequence the background jobs. This interface is a node in one of the four foreground communication channels.

The foreground processor directs background computation through a set of background processor directives. These directives are transmitted over the communication channel as a function call with a background processor node address. The function call consists of a 32 bit primary code and an optional 32 bit parameter. The call addressee may respond with a 32 bit parameter. Organization of the primary code is as follows.

4 bits) Node address
4 bits) Unassigned
8 bits) Function code
16 bits) Interrupt address

Function codes are translated as follows

00) Null function
01) Read status register (response)
02) Read program address (response)
03) Interrupt and idle
04) Enter base address (parameter)
05) Enter limit address (parameter)
06) Enter status data (parameter)
07) Enter program address and start (parameter)

All function calls to the background processor node store the interrupt address in a holding register. This includes the null function call. The address in the holding register is used to interrupt the foreground processor when required by conditions in the background processor status register.

The interrupt process is accomplished by responding to a general channel call with the address in the holding register. This address begins a foreground processor sequence to process the interrupt request.

Individual function calls are described as follows.

---- Function one - Read status register

This function call directs the background processor channel node to respond with a readout of the processor status register. The status register data consists of 16 bits which are transmitted in the low order portion of the 32 bit response field. See 'Background Processor Status Register' for bit significance in this data.

---- Function two - Read program address

This function call directs the background processor channel node to respond with a readout of the current value in the program address register. The background processor is not stopped in this process.

---- Function three - Interrupt and idle

This function call directs the background processor channel node to interrupt the background processor computation and change to an idle mode. No further activity will occur in this processor until directed by another foreground call.

---- Function four - Enter base address

This function call directs the background processor channel node to replace the current value of the base address with the value in the function parameter. The background processor should be in an idle mode for this call.

---- Function five - Enter limit address

This function call directs the background processor channel node to replace the current value of the limit address with the value in the function parameter. The background processor should be in an idle mode for this call.

---- Function six - Enter status data

This function call directs the background processor channel node to enter the status register with the lowest order 16 bits of the parameter data. See 'Background Processor Status Register' for details of this register.

---- Function seven - Enter program address and start

This function call directs the background processor channel node to enter the program address register with the function parameter data and then begin background processor execution of the instruction code at that common memory address.

Instruction Summary

```
000---  Error exit
001---  Normal exit

002i-k  Jump to parcel Ak and hold return parcel Ai
003---  Jump to constant parcel

004--k  Jump to constant parcel if semaphore Ak clear
005--k  Jump to constant parcel if semaphore Ak set

006--k  Jump to constant parcel if semaphore Ak clear.  Set semaphore
007--k  Jump to constant parcel if semaphore Ak set.  Clear semaphore

010--k  Jump to constant parcel if Ak is zero
011--k  Jump to constant parcel if Ak is nonzero

012--k  Jump to constant parcel if Ak is positive
013--k  Jump to constant parcel if Ak is negative

014--k  Jump to constant parcel if Sk is zero
015--k  Jump to constant parcel if Sk is nonzero

016--k  Jump to constant parcel if Sk is positive
017--k  Jump to constant parcel if Sk is negative

020ijk  Enter Ai with integer sum Aj + Ak
021ijk  Enter Ai with integer difference Aj - Ak

022ijk  Enter Ai with integer product Aj * Ak
023---  Unassigned

024i-k  Enter Ai from Sk
025i--  Enter Ai from L

026ijk  Enter Ai with positive jk
027ijk  Enter Ai with negative jk

030--k  Enter M bits where Vk has zero elements
031--k  Enter M bits where Vk has nonzero elements

032--k  Enter M bits where Vk has positive elements
033--k  Enter M bits where Vk has negative elements

034--k  Enter M from Sk
035--k  Alter status register enable flags

036--k  Enter L from Ak
037---  Unassigned
```

```
040i--  Enter Ai with positive 16 bit constant
041i--  Enter Ai with negative 16 bit constant

042i--  Enter Ai with 32 bit constant
043---  Unassigned

044i--  Enter Ai from constant local memory address
045i--  Store Ai into constant local memory address

046i-k  Enter Ai from local memory address Ak
047i-k  Store Ai into local memory address Ak

050i--  Enter Si with positive 16 bit constant
051i--  Enter Si with negative 16 bit constant

052i--  Enter Si with positive 32 bit constant
053i--  Enter Si with 64 bit constant

054i--  Enter Si from constant local memory address
055i--  Store Si into constant local memory address

056i-k  Enter Si from local memory address Ak
057i-k  Store Si into local memory address Ak

060ijk  Enter Si from common memory address Aj + Ak
061ijk  Store Si into common memory address Aj + Ak

062---  Unassigned
063---  Unassigned

064---  Unassigned
065---  Unassigned

066---  Unassigned
067---  Unassigned

070ijk  Enter Vi from common memory address Aj with increment Ak
071ijk  Store Vi into common memory address Aj with increment Ak

072ijk  Enter Vi from common memory address Aj + Vk
073ijk  Store Vi into common memory address Aj + Vk

074i-k  Enter Vi from local memory address Ak
075i-k  Store Vi into local memory address Ak

076---  Unassigned
077---  Unassigned
```

100ijk Enter Si with logical product Sj and Sk
101ijk Enter Si with logical product Sj and complement Sk

102ijk Enter Si with logical difference Sj and Sk
103ijk Enter Si with logical sum Sj and Sk

104ijk Enter Si with integer sum Sj + Sk
105ijk Enter Si with integer difference Sj - Sk

106i-k Enter Si with population count of Sk
107i-k Enter Si with leading zero count of Sk

110ijk Enter Si with Si shifted left 64 - jk
111ijk Enter Si with Si shifted right jk

112ijk Enter Si with Si,Sj shifted left Ak
113ijk Enter Si with Sj,Si shifted right Ak

114i-- Enter Si from M
115i-- Enter Si with real time count

116ijk Enter Si with positive jk
117ijk Enter Si with negative jk

120ijk Enter Si with floating sum Sj + Sk
121ijk Enter Si with floating difference Sj - Sk

122i-k Enter Si with integer form of floating Sk
123--- Unassigned

124ijk Enter Si with floating product Sj * Sk
125ijk Enter Si with reciprocal iteration 2 - Sj * Sk

126i-k Enter Si with reciprocal approximation of Sk
127--- Unassigned

130i-k Enter Si with zero extended Ak
131i-k Enter Si with sign extended Ak

132i-k Enter Si with floating form of Ak
133--- Unassigned

134--- Unassigned
135--- Unassigned

136--- Unassigned
137--- Unassigned

140ijk Enter Vi with logical product Sj and Vk
141ijk Enter Vi with logical product Vj and Vk

142ijk Enter Vi with logical difference Sj and Vk
143ijk Enter Vi with logical difference Vj and Vk

144ijk Enter Vi with logical sum Sj and Vk
145ijk Enter Vi with logical sum Vj and Vk

146ijk Enter Vi with Sj masked into Vk
147ijk Enter Vi with Vj masked into Vk

150ijk Enter Vi with Vj elements shifted left Ak
151ijk Enter Vi with Vj elements shifted right Ak

152ijk Enter Vi with Vj long shifted left Ak
153ijk Enter Vi with Vj long shifted right Ak

154ijk Enter Vi with floating product Sj * Vk
155ijk Enter Vi with floating product Vj * Vk

156ijk Enter Vi with reciprocal iteration 2 - Vj * Vk
157i-k Enter Vi with reciprocal approximation Vk

160ijk Enter Vi with integer sum Sj + Vk
161ijk Enter Vi with integer sum Vj + Vk

162ijk Enter Vi with integer difference Sj - Vk
163ijk Enter Vi with integer difference Vj - Vk

164i-k Enter Vi with population count of Vk
165i-k Enter Vi with leading zero count of Vk

166--- Unassigned
167--- Unassigned

170ijk Enter Vi with floating sum Sj + Vk
171ijk Enter Vi with floating sum Vj + Vk

172ijk Enter Vi with floating difference Sj - Vk
173ijk Enter Vi with floating difference Vj - Vk

174i-k Enter Vi with integer form of floating Vk
175--- Unassigned

176--- Unassigned
177--- Pass

CRAY RESEARCH, INC.

CRAY X-MP COMPUTER SYSTEMS

CRAY X-MP SERIES
MAINFRAME REFERENCE MANUAL

HR-0032

CRAY RESEARCH, INC.

CRAY X-MP COMPUTER SYSTEMS

CRAY X-MP SERIES MAINFRAME REFERENCE MANUAL

HR-0032

Copyright © 1982 by CRAY RESEARCH, INC. This manual or parts thereof may not be reproduced in any form without permission of CRAY RESEARCH, INC.

RECORD OF REVISION

PUBLICATION NUMBER HR-0032

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	November, 1982 - Original printing.

PREFACE

This publication describes the functions of the CRAY X-MP Series of Computer Systems. It is written to assist programmers and engineers and assumes a familiarity with digital computers.

This manual describes the overall computer system, its configurations, and its equipment. It also describes the operation of the Central Processing Units, which execute instructions, provide memory protection, report hardware exceptions, and provide interprocessor communications within the CRAY X-MP Series of Computer Systems.

Details of the CRAY I/O Subsystem, the DD-29 Disk Storage Unit, and the Solid-state Storage Device are given in the following publications:

HR-0030	CRAY I/O Subsystem Hardware Reference Manual
HR-0630	Mass Storage Subsystem Hardware Reference Manual
HR-0031	Solid-state Storage Device (SSD) Reference Manual

//

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

//

CONTENTS

PREFACE iii

1. SYSTEM DESCRIPTION 1-1

INTRODUCTION 1-1
CONVENTIONS 1-4
Italics 1-4
Register conventions 1-4
Number conventions 1-4
Clock period 1-4
SYSTEM COMPONENTS 1-5
Central Processing Units 1-5
I/O Subsystem 1-8
DD-29 Disk Storage Unit 1-9
Solid-state Storage Device 1-11
Condensing units 1-12
Power distribution units 1-13
Motor-generators 1-14
INTERFACES 1-15
SYSTEM CONFIGURATION 1-15
CPU I/O 1-17
Solid-state Storage Device 1-18
I/O Subsystem 1-18
Interfaces to front-end computer 1-19
System deadstart 1-20

2. CPU SHARED RESOURCES 2-1

INTRODUCTION 2-1
CENTRAL MEMORY 2-1
Memory organization 2-2
Memory addressing 2-3
Memory access 2-3
Conflict resolution 2-5
Bank Busy conflict 2-5
Simultaneous Bank conflict 2-5
Section conflict 2-5
Memory access priorities 2-5
16-bank phasing 2-6
Memory error correction 2-6

2. CPU SHARED RESOURCES (continued)

INTER-CPU COMMUNICATION SECTION 2-8
Real-time clock 2-9
Inter-CPU communication and control 2-10
Shared Address and Shared Scalar registers 2-10
Semaphore registers 2-11
CPU INPUT/OUTPUT SECTION 2-12
Data transfer for Solid-state Storage Device (SSD) 2-14
Data transfer for I/O Subsystem 2-14
6 Mbytes per second channels 2-14
Multi-CPU programming 2-15
6 Mbytes per second channel operation 2-16
Input channel programming 2-17
Input channel error conditions 2-18
Output channel programming 2-19
Programmed master clear to external device 2-19
Memory access 2-20
I/O lockout 2-20
Memory bank conflicts 2-20
I/O memory conflicts 2-23
I/O memory request conditions 2-23
I/O memory addressing 2-23

3. CPU CONTROL SECTION 3-1

INTRODUCTION 3-1
INSTRUCTION ISSUE AND CONTROL 3-1
Program Address register 3-2
Next Instruction Parcel register 3-2
Current Instruction Parcel register 3-2
Lower Instruction Parcel register 3-3
Instruction buffers 3-3
EXCHANGE MECHANISM 3-5
Exchange package 3-5
Processor number 3-7
Memory error data 3-7
Exchange registers 3-8
Exchange Address register 3-8
Mode register 3-8
Flag register 3-10
Cluster Number register 3-11
Program State register 3-11
A registers 3-12
S registers 3-12
Program Address register 3-12
Active exchange package 3-12

3. CPU CONTROL SECTION (continued)

Exchange sequence 3-12
Exchange initiated by deadstart sequence 3-13
Exchange initiated by interrupt flag set 3-13
Exchange initiated by program exit 3-13
Exchange sequence issue conditions 3-14
Exchange package management 3-14
MEMORY FIELD PROTECTION 3-15
Instruction Base Address register 3-16
Instruction Limit Address register 3-16
Data Base Address register 3-17
Data Limit Address register 3-17
Program range error 3-17
Operand range error 3-18
PROGRAMMABLE CLOCK 3-18
Instructions 3-18
Interrupt Interval register 3-18
Interrupt Countdown counter 3-19
Clear programmable clock interrupt request 3-19
DEADSTART SEQUENCE 3-19

4. CPU COMPUTATION SECTION 4-1

INTRODUCTION 4-1
OPERATING REGISTERS 4-3
ADDRESS REGISTERS 4-3
A registers 4-3
B registers 4-5
SCALAR REGISTERS 4-6
S registers 4-6
T registers 4-9
VECTOR REGISTERS 4-9
V registers 4-9
V register reservations and chaining 4-12
Vector control registers 4-13
Vector Length register 4-13
Vector Mask register 4-13
FUNCTIONAL UNITS 4-14
Address functional units 4-14
Address Add functional unit 4-15
Address Multiply functional unit 4-15
Scalar functional units 4-15
Scalar Add functional unit 4-15
Scalar Shift functional unit 4-16
Scalar Logical functional unit 4-16
Scalar Population/Parity/Leading Zero functional unit 4-16

4. CPU COMPUTATION SECTION (continued)

Vector functional units 4-16
Vector functional unit reservation 4-17
Vector Add functional unit 4-17
Vector Shift functional unit 4-17
Vector Logical functional unit 4-18
Vector Population/Parity functional unit 4-18
Floating-point functional units 4-18
Floating-point Add functional unit 4-19
Floating-point Multiply functional unit 4-19
Reciprocal Approximation functional unit 4-19
ARITHMETIC OPERATIONS 4-20
Integer arithmetic 4-20
Floating-point arithmetic 4-21
Normalized floating-point numbers 4-22
Floating-point range errors 4-22
Floating-point Add functional unit 4-22
Floating-point Multiply functional unit 4-23
Floating-point Reciprocal Approximation functional unit 4-24
Double-precision numbers 4-24
Addition algorithm 4-25
Multiplication algorithm 4-25
Division algorithm 4-26
Newton's method 4-28
Derivation of the division algorithm 4-28
LOGICAL OPERATIONS 4-32

5. CPU INSTRUCTIONS 5-1

INSTRUCTION FORMAT 5-1
1-parcel instruction format with discrete *j* and *k* fields 5-1
1-parcel instruction format with combined *j* and *k* fields 5-2
2-parcel instruction format with combined *j*, *k*, and *m* fields 5-2
2-parcel instruction format with combined *i*, *j*, *k*, and *m* fields 5-3
SPECIAL REGISTER VALUES 5-4
INSTRUCTION ISSUE 5-5
INSTRUCTION DESCRIPTIONS 5-5

APPENDIX SECTION

A. INSTRUCTION SUMMARY A-1

APPENDIX SECTION (continued)

B. 6 MBYTES PER SECOND CHANNEL DESCRIPTIONS B-1

INTRODUCTION B-1
6 MBYTES PER SECOND INPUT CHANNEL SIGNAL SEQUENCE B-1
Data bits 2^0 through 2^{15} B-1
Parity bits 0 through 3 . . . . . B-2
Ready signal B-3
Resume signal B-3
Disconnect signal B-3
6 MBYTES PER SECOND OUTPUT CHANNEL SIGNAL SEQUENCE B-3
Data bits 2^0 through 2^{15} B-4
Parity bits 0 through 3 . . . . . B-5
Ready signal B-5
Resume signal B-5
Disconnect signal B-5

INDEX

FIGURES

1-1 CRAY X-MP mainframe with a Cray I/O Subsystem and an SSD . . . 1-2
1-2 Basic organization of the CRAY X-MP CPUs 1-5
1-3 Control and data paths for a CPU 1-6
1-4 CRAY X-MP mainframe chassis 1-7
1-5 I/O Subsystem chassis 1-8
1-6 DD-29 Disk Storage Unit 1-9
1-7 Solid-state Storage Device chassis 1-11
1-8 Condensing unit 1-12
1-9 Power distribution units 1-13
1-10 Motor-generator equipment 1-14
1-11 Typical interface cabinet 1-15
1-12 Block diagram of CRAY X-MP system with increased disk capacity 1-16
1-13 Block diagram of CRAY X-MP system with block multiplexer channels 1-17
2-1 Central Memory organization 2-2
2-2 Memory address (32 banks) 2-3
2-3 Memory address (16 banks) 2-3
2-4 Memory data path with SECDED 2-6
2-5 Error correction matrix 2-8
2-6 Shared registers 2-9
2-7 Basic I/O program flowchart 2-18
2-8 Channel I/O control 2-21
2-9 Input/output data paths 2-22
3-1 Instruction issue and control elements 3-1
3-2 Instruction buffers 3-3
3-3 Exchange package 3-6

FIGURES (continued)

4-1 Address registers and functional units 4-4
4-2 Scalar registers and functional units 4-7
4-3 Vector registers and functional units 4-10
4-4 Integer data formats 4-20
4-5 Floating-point data format 4-21
4-6 Integer multiply in Floating-point Multiply functional unit 4-24
4-7 49-bit floating-point addition 4-25
4-8 Floating-point multiply partial-product sums pyramid 4-27
4-9 Newton's method 4-28
5-1 General form for instructions 5-1
5-2 1-parcel instruction format with discrete *j* and *k* fields . . . 5-2
5-3 1-parcel instruction format with combined *j* and *k* fields 5-2
5-4 2-parcel instruction format with combined *j*, *k*, and *m* fields 5-3
5-5 2-parcel instruction format with combined *i*, *j*, *k*, and *m* fields 5-4
5-6 Vector left double shift, first element, VL greater than 1 . . 5-68
5-7 Vector left double shift, second element,VL greater than 2 . . 5-68
5-8 Vector left double shift, last element 5-68
5-9 Vector right double shift, first element 5-69
5-10 Vector right double shift, second element, VL greater than 1 5-70
5-11 Vector right double shift, last operation 5-70

TABLES

1-1 CRAY X-MP System characteristics 1-2
1-2 DD-29 DSU operational characteristics 1-10
2-1 Access conflicts to shared registers 2-11
2-2 Channel word assembly/disassembly 2-17
B-1 Input channel signal exchange B-2
B-2 Output channel signal exchange B-4

SYSTEM DESCRIPTION 1

INTRODUCTION

The CRAY X-MP Series of Computer Systems are powerful, general purpose multiprocessor systems with two Central Processing Units (CPUs) in each mainframe. Extremely high multiprocessing rates are achieved by combining scalar and vector capabilities of the CPUs, which are joined to a large, fast random-access solid-state memory (RAM) and shared registers.

Vector processing is the performance of iterative operations on sets of ordered data. 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. When two or more vector operations are chained together, two or more results can be produced each 9.5-nanosecond clock period, greatly exceeding the result rates of conventional scalar processing. Scalar operations complement the vector capability by providing solutions to problems not readily adaptable to vector techniques.

The CRAY X-MP Series of Computer Systems has a high performance level, which is considerably beyond that of the CRAY-1 S Series of Computer Systems. Several equipment options allow an optimum CRAY X-MP Computer System to be configured for a particular use. Central Memory of the CRAY X-MP mainframe can be either 2 million 64-bit words (Model 22) or 4 million 64-bit words (Model 24). The mainframe is compatible with all existing models of the CRAY I/O Subsystem and its associated mass storage subsystem. In addition, an optional high performance Solid-state Storage Device (SSD) can be attached to the mainframe. Several combinations of memory size and I/O capabilities are also possible. Figure 1-1 illustrates a CRAY X-MP mainframe with a Cray I/O Subsystem and an SSD.

The CRAY X-MP can also execute CRAY-1 S CFT-generated object code without modification, unless the vector recursion feature is used in CRAY-1 S programs.† Also, certain sequences of CAL instructions involving block memory transfers require explicit synchronization of memory transfers to execute correctly on the CRAY X-MP.

This section describes system components and configurations. Table 1-1 gives overall system characteristics.

† For example, the assembly language instruction V3 V3+V5 produces different results in register V3 when executed by the CRAY-1 and the CRAY X-MP.

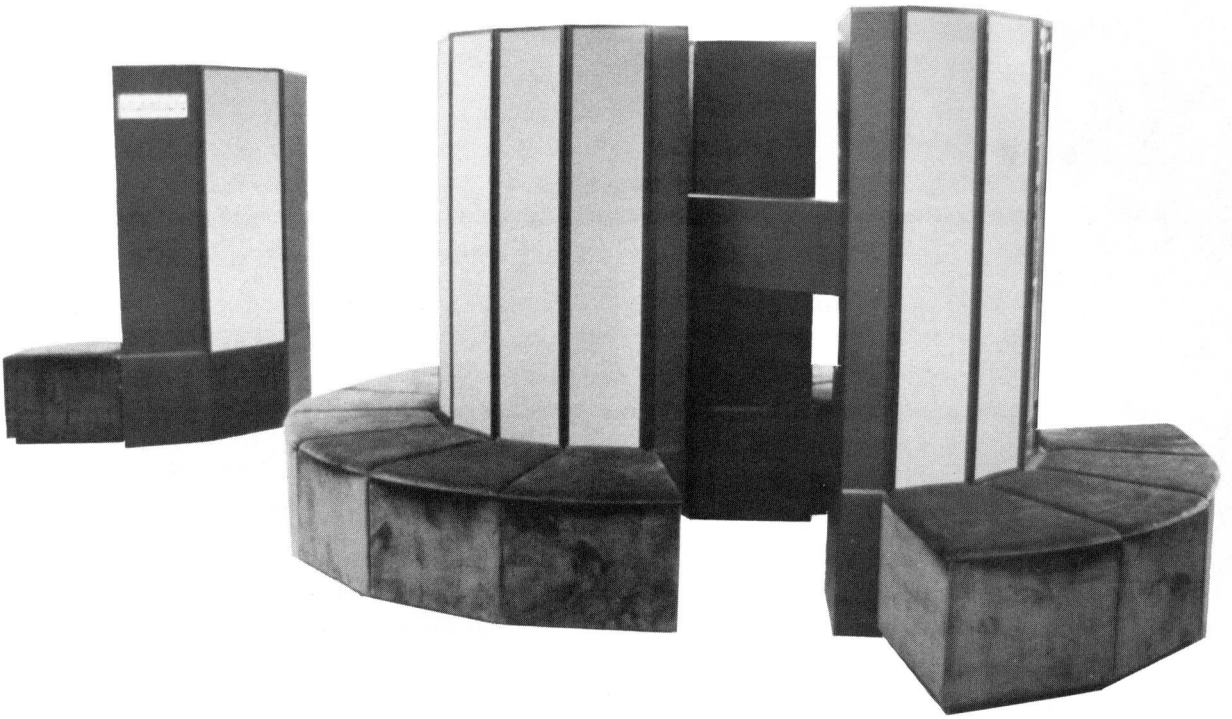


Figure 1-1. CRAY X-MP mainframe with a Cray I/O Subsystem and an SSD

Table 1-1. CRAY X-MP system characteristics

Configuration	- Mainframe with 2 Central Processing Units (CPUs) - I/O Subsystem with 2, 3, or 4 I/O Processors - Optional Solid-state Storage Device (SSD)
CPU speed	- 9.5 ns CPU clock period - 105 million floating-point additions per second per CPU - 105 million floating-point multiplications per second per CPU - 105 million half-precision floating-point divisions per second per CPU - 33 million full-precision floating-point divisions per second per CPU - Simultaneous floating-point addition, multiplication, and reciprocal approximation within each CPU

Table 1-1. CRAY X-MP system characteristics (continued)

Memories	- Up to 4 million 64-bit words in mainframe Central Memory - 65,536 16-bit parcels in Local Memory of each I/O Processor of the I/O Subsystem - 6 direct memory access (DMA) ports to Local Memory (each I/O Processor) - 1, 4, or 8 million 64-bit words of I/O Subsystem Buffer Memory - 8, 16, or 32 million words of SSD memory
Mass storage	- 600 million byte disk drive - 48 disk drives maximum for I/O Subsystem - 35.4 Mbits per second disk drive transfer rate
Input/Output	- One 1250 Mbytes per second Solid-state Storage Device (SSD) channel on mainframe - Two 100 Mbytes per second channels between mainframe and I/O Subsystem for a system with an SSD - Four 100 Mbytes per second channels between mainframe and I/O Subsystem for a system without an SSD - Four 6 Mbytes per second channels - 40 channels; input or output, 24 of which share the six DMA ports per I/O Processor - Mainframe interfaces to I/O Subsystem
Physical	- 45 sq ft floor space for mainframe - 15 sq ft floor space for I/O Subsystem - 15 sq ft floor space for SSD - 5.25 tons, mainframe weight - 1.5 tons, I/O Subsystem weight - 1.5 tons, SSD weight - Liquid refrigeration of each chassis - 400 Hz power from motor-generators

CONVENTIONS

The following conventions are used in this manual.

ITALICS

Italicized lowercase letters, such as *jk*, indicate variable information.

REGISTER CONVENTIONS

Parenthesized register names are used frequently in this manual as a form of 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."

Designations for the A, B, S, T, and V registers are used extensively. For example, "Transmit (T*jk*) to S*i*" means "Transmit the contents of the T register specified by the *jk* designators to the S register specified by the *i* designator."

Register bits are numbered right to left as powers of 2, starting with 2^0. Bit 2^{63} of an S, V, or T register value represents the most significant bit. Bit 2^{23} of an A or B register value represents the most significant bit. (A and B registers are 24 bits.) The numbering conventions for the Exchange Package and the Vector Mask register are exceptions. Bits in the Exchange Package are numbered from left to right and are not numbered as powers of 2 but as bits 0 through 63 with 0 as the most significant and 63 as the least significant. The Vector Mask register has 64 bits, each corresponding to a word element in a vector register. Bit 2^{63} corresponds to element 0, bit 2^0 corresponds to element 63.

NUMBER CONVENTIONS

Unless otherwise indicated, numbers in this manual are decimal numbers. Octal numbers are indicated with an 8 subscript. Exceptions are register numbers, channel numbers, instruction parcels in instruction buffers, and instruction forms given in octal without the subscript.

CLOCK PERIOD

The basic unit of CPU computation time is 9.5 nanoseconds (ns) and is referred to as a clock period (CP). Instruction issue, memory references, and other timing considerations are often measured in CPs.

SYSTEM COMPONENTS

The CRAY X-MP Computer System is composed of a CRAY X-MP mainframe with two CPUs and an I/O Subsystem. Mass storage devices and optional tape devices are also integral parts of a CRAY X-MP Computer System. Optionally, A Cray Research SSD can be a component of a CRAY X-MP Computer System. Supporting this equipment are condensing units for refrigeration, power distribution units for the mainframe, I/O Subsystem, and SSD, and motor-generators providing system power. The system components are described on the following pages.

CENTRAL PROCESSING UNITS

Each CPU has an independent control section and computation section. Both CPUs share Central Memory and the inter-CPU communication and I/O sections. (CPU sections are described in later sections of this publication.) Figure 1-2 represents the basic organization of the two CPUs; figure 1-3 illustrates the components and control and data paths of one CPU in the system.

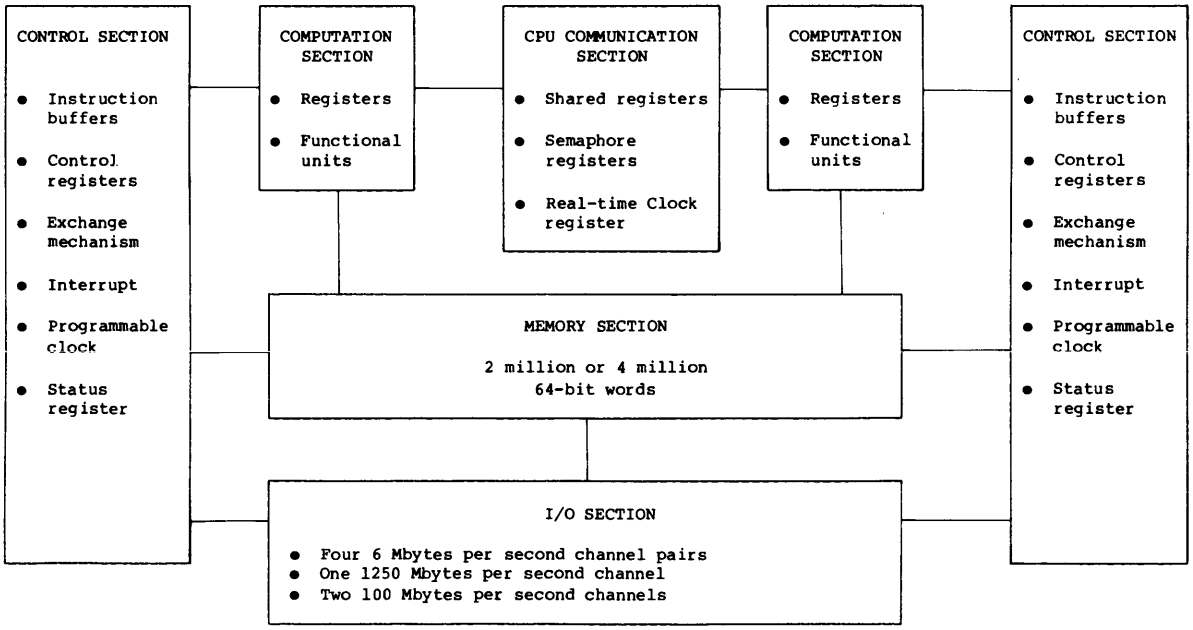


Figure 1-2. Basic organization of the CRAY X-MP CPUs

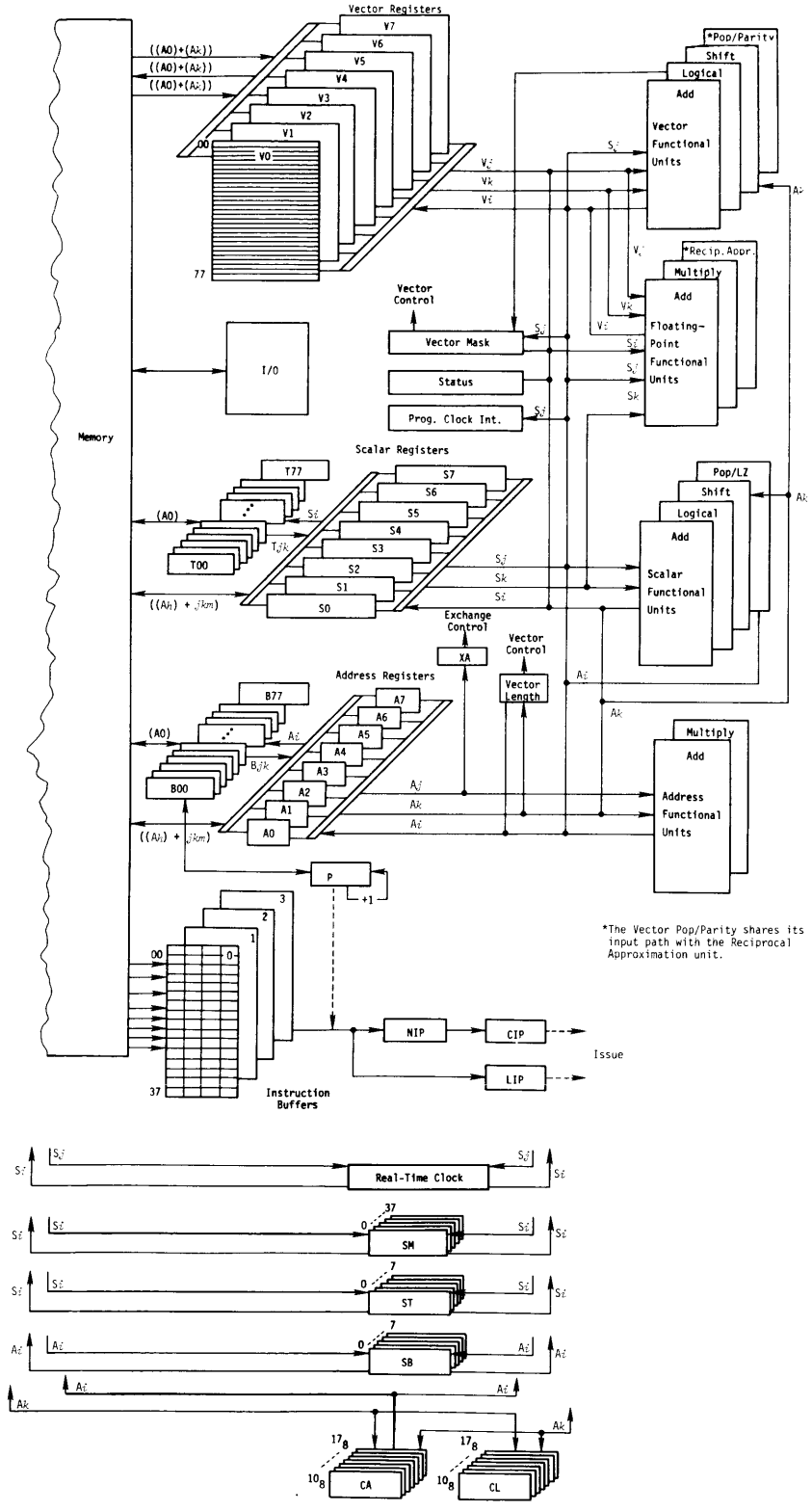


Figure 1-3. Control and data paths for a CPU

The two CPUs are located in the central four columns of the CRAY X-MP mainframe chassis: one in the upper half and one in the lower half. The additional eight columns contain Central Memory. All CRAY X-MP Computer Systems use a 12-column chassis (figure 1-4) with eight columns of memory and four columns of CPUs. For 16-bank machines (two million words), only four memory columns are required for memory and the other four columns are wired for later upgrading to 32 banks (four million words).

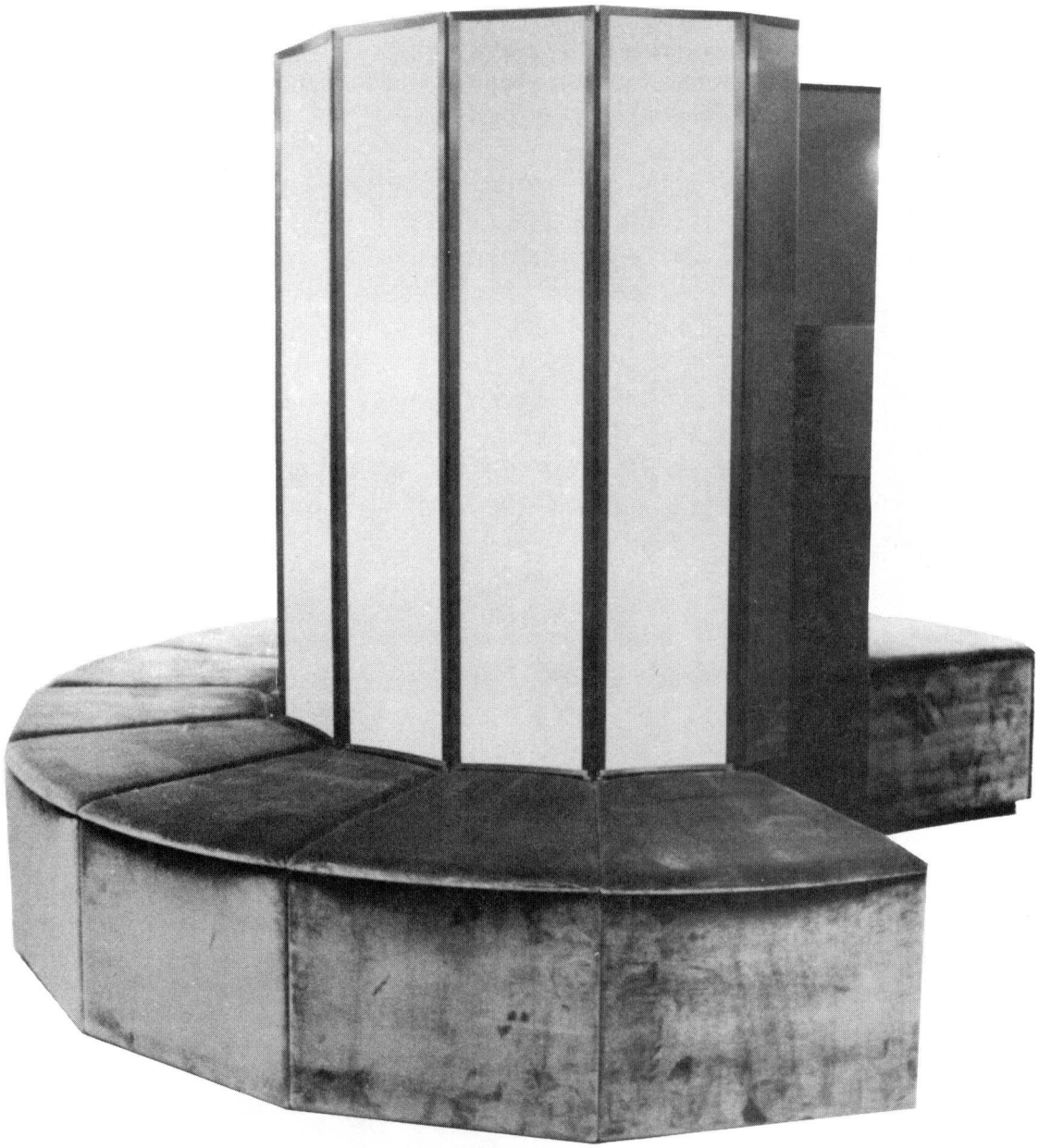


Figure 1-4. CRAY X-MP mainframe chassis

I/O SUBSYSTEM

CRAY X-MP computers are equipped with an I/O Subsystem composed of two, three, or four I/O Processors (IOPs), Buffer Memory, and required interfaces. The I/O Subsystem is designed for fast data transfer between front-end computers, peripheral devices, storage devices, and Buffer Memory or between Buffer Memory and Central Memory of a CRAY X-MP mainframe.

Each IOP has a memory section, a control section, a computation section, and an input/output section. I/O sections are independent and handle some portion of the I/O requirements for the system. The I/O Subsystem is housed in a 4-column chassis (figure 1-5). Refer to the CRAY I/O Subsystem Reference Manual, publication HR-0030, for a detailed description of the I/O Subsystem.

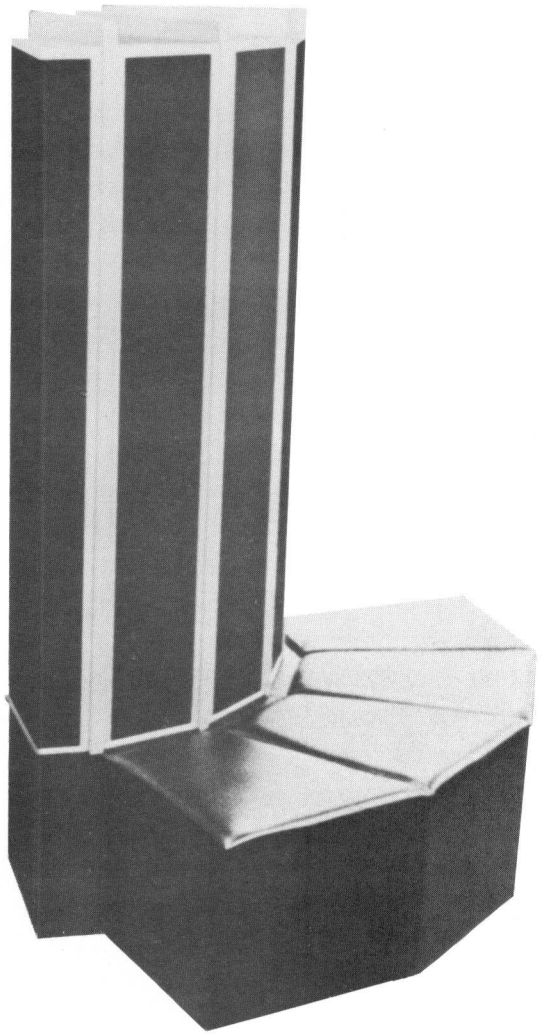


Figure 1-5. I/O Subsystem chassis

DD-29 DISK STORAGE UNIT

The CRAY X-MP Series of Computer Systems uses the DD-29 Disk Storage Unit (DSU) for mass storage. This is a 606 Mbytes disk drive having a data transfer rate of 35.4 Mbits per second.

Up to four DD-29s can be connected to one DCU-4 Disk Controller Unit. The DCU-4 Controller Unit interfaces the disk units with an I/O Processor of an I/O Subsystem through one direct memory access (DMA) port. The I/O Processor and the disk controller unit can transfer data between the DMA port and four DSUs with all DSUs operating at full speed without missing data or skipping revolutions. A minimum of 2 and a maximum of 48 DD-29s can be configured. Figure 1-6 shows a DD-29 DSU. The DCU-4 Disk Controller Unit is housed in the I/O Subsystem chassis.

Each DD-29 DSU has two accesses for connecting it to controllers. The second independent data path to each DSU exists through another Cray Research, Inc., controller. Reservation logic provides controlled access to each DSU. Dynamic sharing of devices is not supported by the Cray Operating System (COS) software. Table 1-2 summarizes DD-29 DSU operational characteristics. Further information about the mass storage subsystem is included in the CRAY I/O Subsystem Reference Manual, CRI publication HR-0030, and the Mass Storage Subsystem Hardware Reference Manual, CRI publication HR-0630.

Figure 1-6. DD-29 Disk Storage Unit

Table 1-2. DD-29 DSU operational characteristics

Bits per drive: 4.848×10^9
Bytes per drive: 606×10^6
Words per drive: 75.8×10^6

Words per sector: 512
Words per track: 9216
Words per logical cylinder: 92,160

Bytes per sector: 4096

Sectors per logical track: 18
Logical tracks per cylinder: 10

Maximum Latency: 16.6 msec.

Access time: 15 - 80 msec.

Transfer rate (maximum):

- One sector: 38.7×10^6 bits per second
- One cylinder (180 sectors): 35.4×10^6 bits per second†
- One drive (823 cylinders): 32.2×10^6 bits per second††

† Rate is less than one sector rate due to the time required to passover the sector address information prerecorded between sectors.

†† Rate is less than one cylinder rate due to the time required to move the heads one track (one cylinder).

SOLID-STATE STORAGE DEVICE

The Solid-state Storage Device (SSD) is used as a device for temporary storage and transfers data between the CRAY X-MP mainframe's Central Memory and the SSD. The speed of these transfers is dependent on the SSD memory size and configuration described in the Solid-state Storage Device (SSD) Reference Manual, CRI publication HR-0031. The maximum speed attained from the SSD to Central Memory is 1250 Mbytes per second. The SSD is housed in a 4-column chassis (figure 1-7).

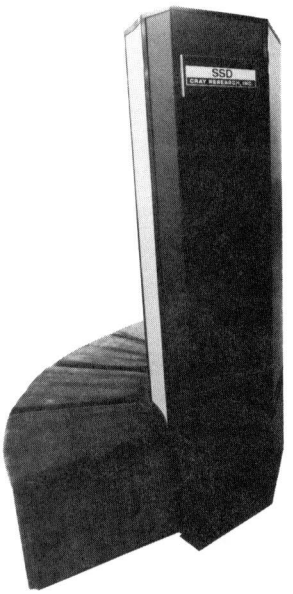


Figure 1-7. Solid-state Storage Device chassis

CONDENSING UNITS

Condensing units (figure 1-8) contain the major components of the refrigeration system used to cool the computer chassis and consists of two 25-ton condensers. Heat is removed from the condensing unit by a second level cooling system that is not part of the CRAY X-MP Computer System. Freon, which cools the computer, picks up heat and transfers it to water in the condensing unit.

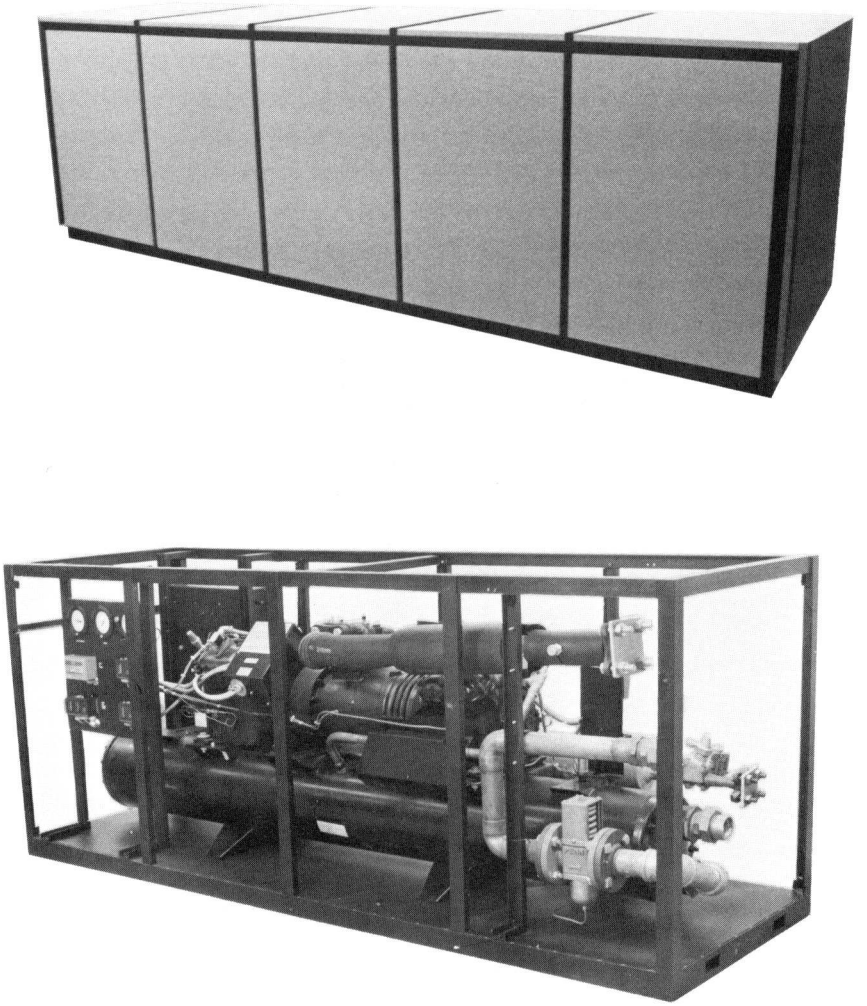


Figure 1-8. Condensing unit

POWER DISTRIBUTION UNITS

The CRAY X-MP mainframe, I/O Subsystem, and SSD all operate from 400 Hz 3-phase power. The mainframe, I/O Subsystem, and SSD have independent power distribution units. The power distribution unit for the mainframe contains adjustable transformers for regulating the voltage to each power supply for the mainframe. The power distribution unit also contains temperature and voltage monitoring equipment that checks temperatures at strategic locations on the mainframe chassis. Automatic warning and shutdown circuitry protects the mainframe in case of overheating or excessive cooling. The control switches for the motor-generators and the condensing unit are mounted on the CRAY X-MP mainframe power distribution unit.

The smaller power distribution unit performs similar functions for the I/O Subsystem chassis or the SSD chassis.

Figure 1-9 shows the power distribution units for the CRAY X-MP mainframe and for the I/O Subsystem or SSD.

Figure 1-9. Power distribution units

MOTOR-GENERATORS

The motor-generator units convert primary power from the commercial power mains to the 400 Hz power used by the CRAY X-MP Computer System. These units isolate the system from transients and fluctuations on the commercial power mains. The equipment consists of two or three motor-generator units and a control cabinet. Figure 1-10 shows a typical motor-generator and the control cabinet.

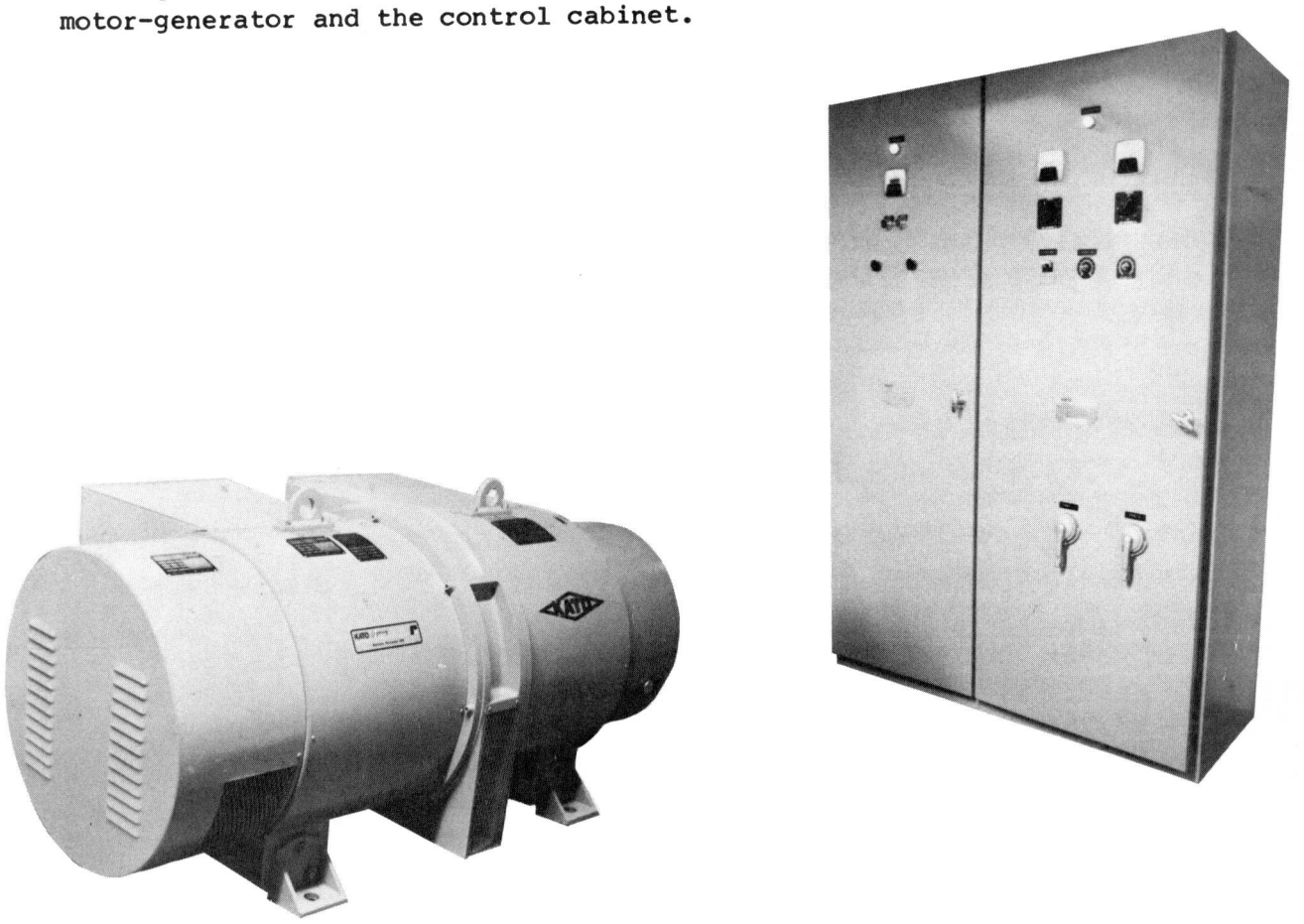


Figure 1-10. Motor-generator equipment

INTERFACES

The CRAY X-MP is designed for use with front-end computers in a computer network. Standard front-end interfaces connect to the Master I/O Processor of a Cray I/O Subsystem via channels with a transfer rate of 6 Mbytes per second.

Each interface is housed in a stand-alone cabinet (figure 1-11) located near the host computer. The cabinet is air cooled and operates directly from the 60 Hz AC power mains. Power consumption and the heat generated by the interface cabinet vary with the complexity of the interface. The cabinet contains two or more logic modules and appropriate cabling connector panels. Internal power supplies provide the required logic and communication voltages. Cabinet grounding is flexible and the unit can be easily integrated into a front-end computer with its specific grounding requirements. The interface uses hardware logic to perform command translation and protocol conversion needed to transfer data. Its operation is invisible to the front-end computer user and the CRAY X-MP user.

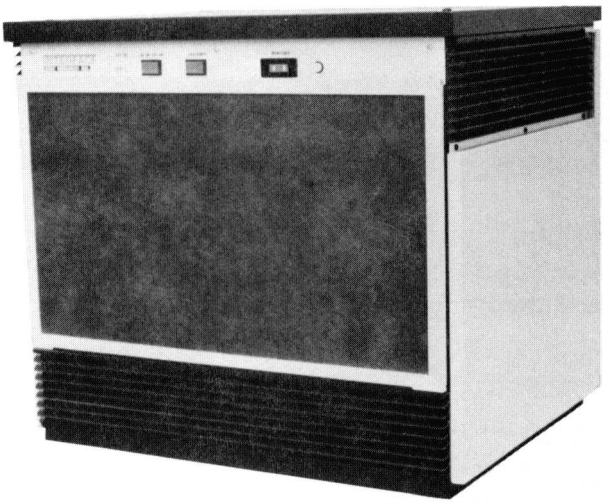


Figure 1-11. Typical interface cabinet

SYSTEM CONFIGURATION

The mainframe of the CRAY X-MP Series of Computer Systems has two Central Processing Units (CPUs). Model 22 has a Central Memory with 2 million 64-bit words (16 banks); model 24 has a Central Memory with 4 million 64-bit words (32 banks). Memory is wired for 4 million words and 32 banks to allow for field upgrade. Figures 1-12 and 1-13 illustrate two

variations of the system configuration. System components can be configured as described in this section. The system deadstart process (system initialization procedure) used to bring the system to an operational state is described later in this section.

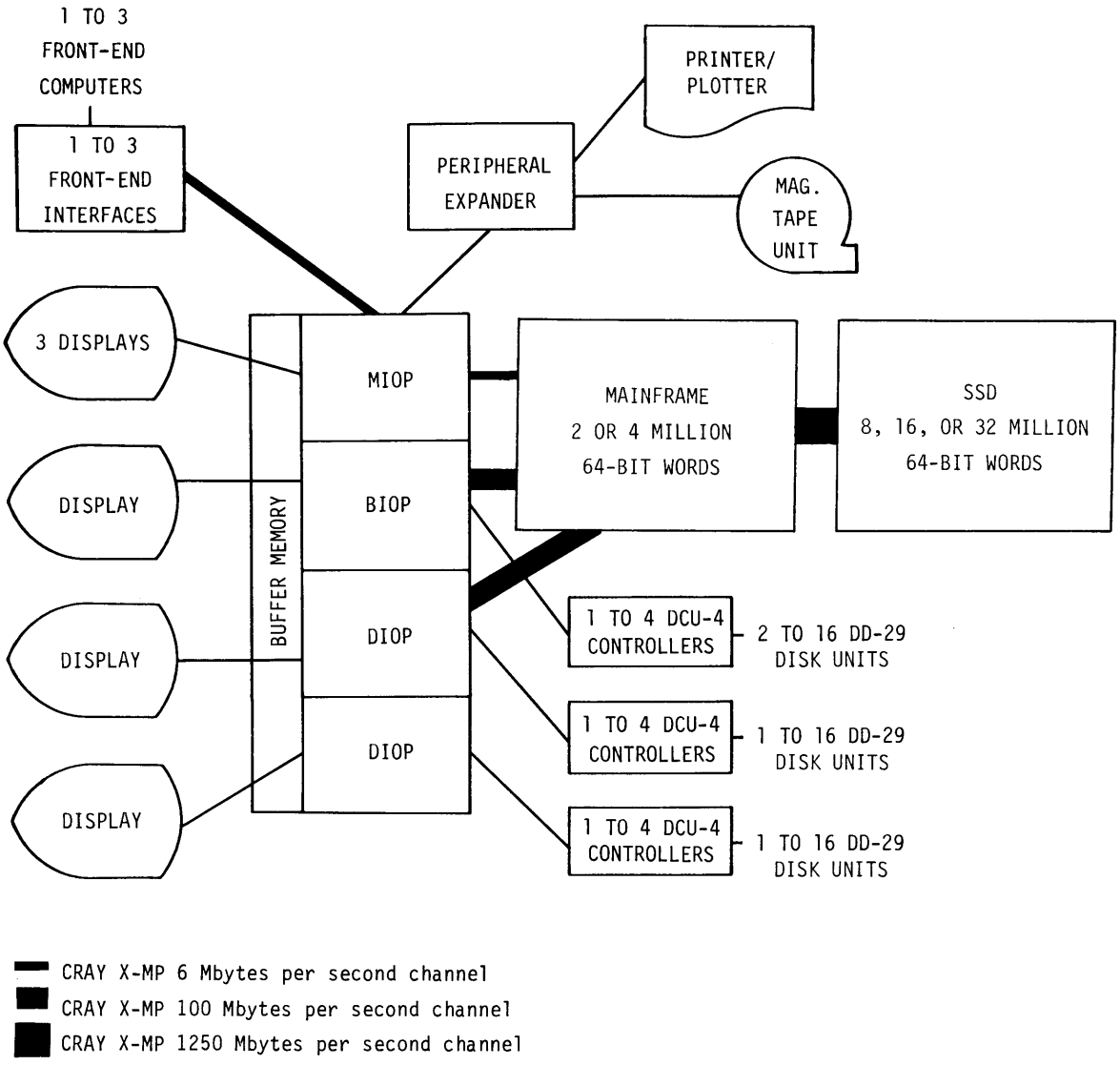


Figure 1-12. Block diagram of CRAY X-MP system with increased disk capacity

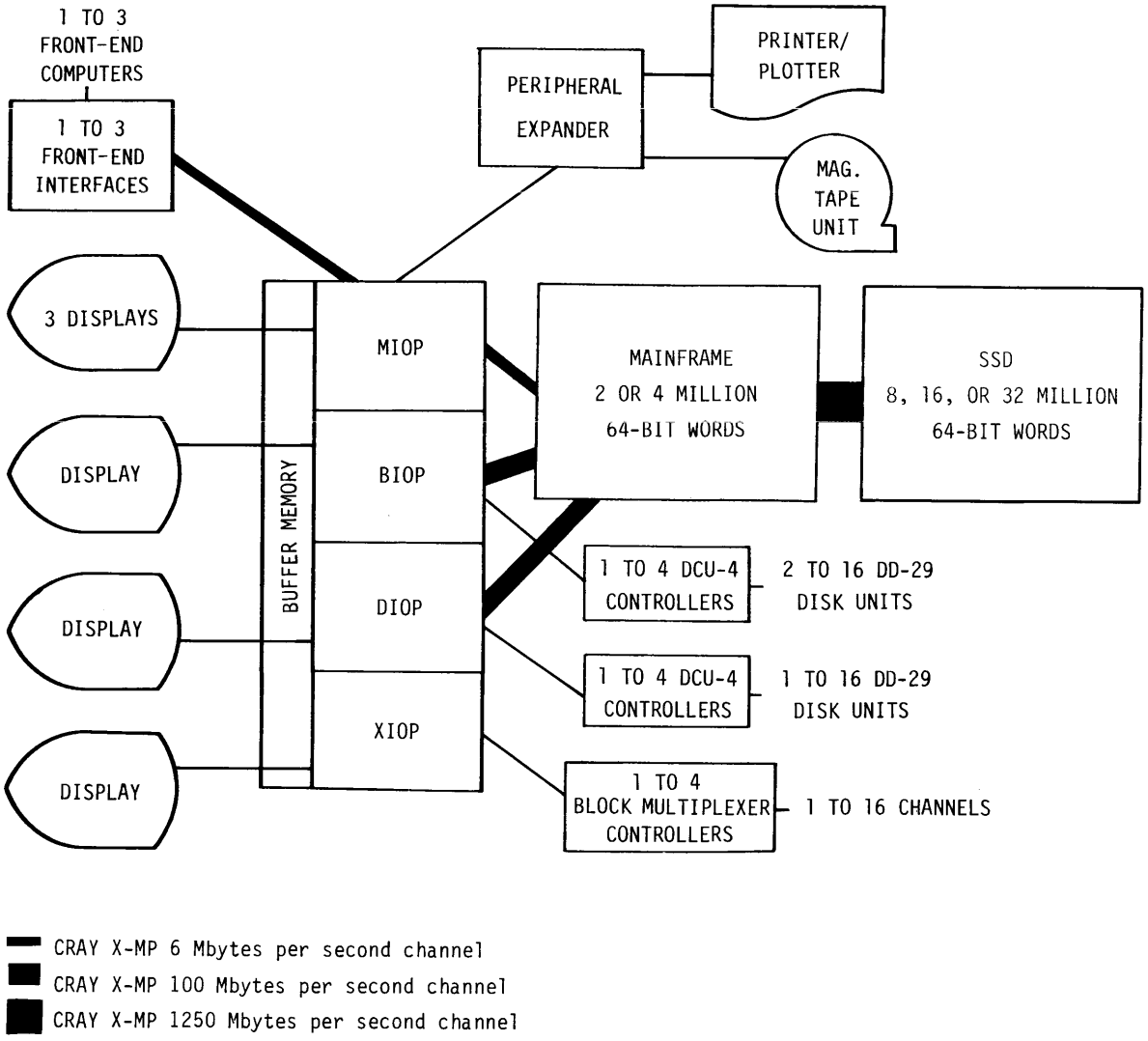


Figure 1-13. Block diagram of CRAY X-MP system with block multiplexer channels

CPU I/O

Extremely high speed data transfers to and from the SSD are achieved using a channel with a transfer rate of 1250 Mbytes per second. High-speed data transfers to and from I/O Processors (IOPs) in the I/O Subsystem are achieved using two channels each with a transfer rate of 100 Mbytes per second. Four I/O channel pairs provide access to front-end computers, mass storage controllers, and to the I/O Subsystem.

SOLID-STATE STORAGE DEVICE

The Solid-state Storage Device (SSD) is an optional high performance device which can be configured with 8, 16, or 32 million 64-bit words of memory. A special CRAY X-MP interface cable set is used. The Solid-state Storage Device (SSD) Reference Manual, CRI publication HR-0031, provides additional information about the SSD.

I/O SUBSYSTEM

The I/O Subsystem is standard on all models in the CRAY X-MP Series of Computers and has two to four I/O Processors (IOPs) and 1, 4, or 8 million 64-bit words of Buffer Memory. The I/O Subsystem in the CRAY X-MP system is identical to the one used in a CRAY-1 S Series system.

The Master I/O Processor (MIOP) controls the front-end interfaces and the standard group of station† peripherals. The Peripheral Expander interfaces the station peripherals to one direct memory access (DMA) port of the MIOP. The MIOP also connects to Buffer Memory and to the mainframe over a 6 Mbytes per second channel. The MIOP communicates with the Cray Operating System (COS) to coordinate the activities of the I/O Subsystem.

The Buffer I/O Processor (BIOP) is the main link between Central Memory and the mass storage devices. Data from mass storage is often transferred through the BIOP's Local Memory to the mainframe's Central Memory via a 100 Mbytes per second channel.

The Disk I/O Processor (DIOP) is used for additional disk storage units. This processor can accommodate up to four disk controller units with up to 16 disk storage units. The DIOP uses one DMA port for each controller, another DMA port to connect to with Buffer Memory, and another DMA port to connect a second 100 Mbytes per second channel to the mainframe Central Memory.

The Auxiliary I/O Processor (XIOP) is used for block multiplexer channels and interfaces to a maximum of four BMC-4 Block Multiplexer Controllers, each of which can handle up to four block multiplexer channels. The XIOP uses one DMA port for each controller and another DMA port to connect with Buffer Memory.

The I/O Subsystem hardware can accommodate two 100 Mbytes per second channels to the mainframe, that is, one channel to the BIOP and another channel to the DIOP or XIOP to transfer data simultaneously between the I/O Subsystem and the mainframe's Central Memory. (Software to support

† The term station means both hardware and software. Station is the link to the front end or can act as a limited front end (as the MIOP).

the 100 Mbytes per second channel to the XIOP is currently not available.) The CRAY X-MP mainframe hardware can support up to two 100 Mbytes per second channels and a 1250 Mbytes per second channel; alternatively, without the 1250 Mbytes per second channel, the mainframe can suppport up to four 100 Mbytes per second channels. Therefore, the CRAY X-MP can be configured with one to four separate I/O Subsystems; however, systems with more than two I/O Subsystems cannot connect an SSD to the CRAY X-MP mainframe.

The CPU input/output section for the CRAY X-MP is described in section 2 of this manual. The CRAY I/O Subsystem Hardware Reference Manual, publication HR-0030, provides additional information on I/O Subsystem communication.

INTERFACES TO FRONT-END COMPUTER

A front-end computer system is self contained and executes under the control of its own operating system. Standard interfaces connect the CRAY X-MP I/O channels to channels of front-end computers, providing input data to the CRAY X-MP Computer System and receiving output from it for distribution to peripheral equipment. Interfaces compensate for differences in channel widths, machine word size, electrical logic levels, and control signals. The Master I/O Processor of the I/O Subsystem communicates with a front-end computer system through a 6 Mbytes per second channel pair to a channel adapter module in the CRAY X-MP mainframe. Communication continues through a front-end interface (figure 1-11) to the front-end computer typically via a front-end computer I/O channel.

A primary goal of the interface is to maximize the use of the front-end channel connected to the CRAY X-MP Computer System. Since the MIOP channel connected to the interface is faster than any front-end channel connected to the interface, the burst rate of the interface is limited by the maximum rate of the front-end channel.

Peripheral equipment attached to the front-end computer varies depending on the use of the CRAY X-MP Computer System.

Interfaces to front-end computers allow the front-end computers to service the CRAY X-MP Computer System in the following ways:

- As a master operator station
- As a local operator station
- As a local batch entry station
- As a data concentrator for multiplexing several other stations into a single CRAY X-MP channel

- As a remote batch entry station
- As an interactive communication station

Detailed information about the front-end system and the front-end communication protocol is beyond the scope of this publication.

SYSTEM DEADSTART

The I/O Subsystem is deadstarted from the Peripheral Expander magnetic tape unit. Subsequent I/O Subsystem deadstarting can be from magnetic tape or a DD-29 disk unit. Once the I/O Subsystem is operating, the CRAY X-MP mainframe can be deadstarted from the Peripheral Expander magnetic tape unit or the DD-29 disk unit. In case of a failure in the MIOP, a maintenance deadstart panel can be used to load a deadstart or diagnostic program into the MIOP or BIOP.

The startup command and procedures for installing deadstart files on the DD-29 disk unit are described in the I/O Subsystem (IOS) Operator's Guide, CRI publication SG-0051.

CPU SHARED RESOURCES 2

INTRODUCTION

The two Central Processing Units (CPUs) share the mainframe's Central Memory, the inter-CPU communication section, and input/output section. These areas common to the CPUs are described in the following pages.

CENTRAL MEMORY

Central Memory consists of 16 or 32 banks of solid-state, random access memory (RAM) and is shared by the CPUs and the I/O section. Two standard Central Memory sizes are available: two million words with 16 banks and four million words with 32 banks. Banks are independent of each other. Each word is 72 bits with 64 data bits and 8 check bits. Sequentially addressed words reside in sequential banks.

Central Memory cycle time is 4 clock periods (CPs) or 38 nanoseconds (ns). Access time, the time required to fetch an operand from Central Memory to an operating register, is 14 CPs (133 ns) for A and S register operands. Access time is 17 CPs + vector length for a vector (V) register and 16 CPs + block length for a block transfer to an intermediate address (B) or intermediate scalar (T) register.

The maximum transfer rate per CPU for B, T, and V registers is three words per CP; for address (A) and scalar (S) registers per CPU, it is one word per 2 CPs. Transfer of instructions to instruction buffers occurs at a rate of 32 parcels (eight words) per CP. For the I/O section, the transfer rate is two words per CP.

Central Memory features are summarized below and are described in detail in the following paragraphs.

- Shared access from the two CPUs
- 2 million or 4 million words of integrated circuit memory
- 64 data bits and 8 error correction bits per word
- 16 or 32 interleaved banks
- 4-CP bank cycle time
- Single error correction/double error detection (SECDED)
- 3 words per CP transfer rate to B, T, and V registers per CPU

- 1 word per 2 CP transfer rate to A and S registers per CPU
- 8 words per CP transfer rate to instruction buffers
- 2 words per CP transfer rate to I/O concurrent with all memory activity except instruction fetch and exchange

MEMORY ORGANIZATION

Central Memory is organized into four sections with four or eight banks in each section. The 16-bank phasing is standard for a 2 million word system, and 32-bank phasing is standard for a 4 million word system.

Each bank occupies one-quarter of a column and contains 18 modules. Each module contributes four data or check bits to each 72-bit word in the bank; a memory word consists of 64 data bits and 8 check bits.

Each CPU is connected to an independent access path into each of the four sections, as shown in figure 2-1. This configuration allows up to eight memory references per clock period.

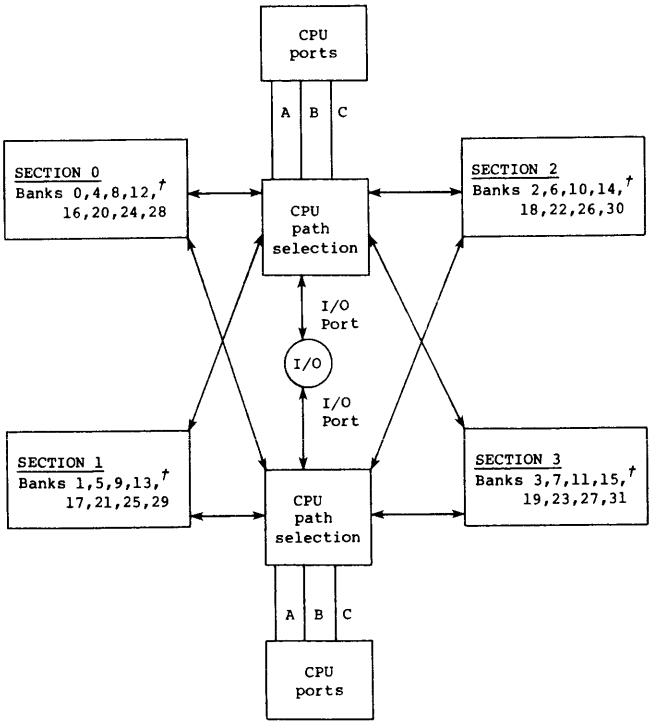


Figure 2-1. Central Memory organization

† Low-numbered 4 banks in each section are for a 16-bank system only; a 32-bank system has 8 banks in each section.

MEMORY ADDRESSING

A word in a 32-bank memory is addressed in a maximum of 22 bits as shown in figure 2-2. The low-order 5 bits specify one of the 32 banks. The next 12-bit field specifies an address within the chip. The high-order 5 bits specify one chip on the module.

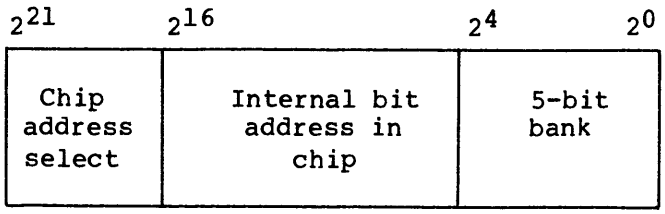


Figure 2-2. Memory address (32 banks)

A word in a 16-bank memory is addressed in a maximum of 21 bits as shown in figure 2-3. In this case, the low-order 4 bits specify one of the 16 banks. The next 12-bit field specifies an address within the chip. The high-order 5 bits specify one chip on the module.

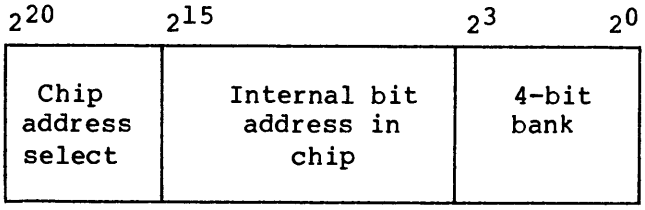


Figure 2-3. Memory address (16 banks)

MEMORY ACCESS

Each CPU in the CRAY X-MP system has four memory access ports, referred to as Port A, Port B, Port C, and I/O. Each port is capable of making one reference per CP. Ports A, B, and C are used for CPU register transfers.

B, T, and vector memory instructions issue to a particular memory port:

Vector read (block reads only), B read instructions (176, 034) use Port A.
Vector read (block reads only), T read instructions (176, 036) use Port B.
Vector store, B, or T store instructions (177, 035, and 037) and scalar instructions (100-137) use Port C.

Once an instruction issues to a port, that port is reserved until all references are made for that instruction.

The references for a block transfer (V,B,T) are made in sequence. However, since each reference is examined individually for possible conflicts, the data flow for a transfer may not be continuous. If an instruction requires a port that is busy, issue is blocked. Total execution time of the transfer depends upon the number and type of conflicts encountered during the transfer.

Because concurrent block reads and writes are not examined for memory overlap hazard conditions (that is, read before write or write before read), the software must detect the cases where this condition occurs and assure sequential operation. The bidirectional memory mode enable (002500), bi-directional memory mode disable (002600), and the complete memory reference (002700) instructions are provided to resolve these cases and assure sequential operation. If the bidirectional memory mode is clear, block reads and writes are not allowed to operate concurrently within that CPU. Instruction 002700 allows the program to wait until the last references of all preceding block transfers are past the conflict resolution stage within the CPU issuing it and the transferred data is being transmitted to the designated memory or register locations. Instruction 002700 provides software a mechanism, wherever necessary in the program, to guarantee sequential memory operation within a CPU or between CPUs.

Issue of scalar memory references requires Ports A, B, and C to be available, ensuring sequential operation between block transfers and scalar references within a CPU. A scalar reference conflict is detected in CP 3 of execution. If a conflict occurs, one more scalar reference is allowed to issue. A third scalar reference holds issue if the conflict condition still exists for the preceding scalar reference. Scalar references always execute in the order they are issued within a CPU. Instruction 002700 detects when all scalar references are past the conflict resolution stage within the CPU issuing it.

One-half of the CPU I/O channels reference memory through each CPU's I/O port. The I/O port can be active regardless of the activities on Ports A, B, or C.

When an instruction fetch request occurs, all referencing from the eight memory ports is inhibited. When memory is quiet (0 to 3 CPs), the fetch proceeds and references 32 banks in the next 4 CPs (6 CPs if 16 banks). Then the referencing of the eight ports is enabled.

An exchange requires all activities within a CPU to complete before the exchange request is made. When the exchange request is made, all referencing from the four memory ports of the other CPU is inhibited. When memory is quiet (0 to 3 CPs), the exchange proceeds and references 16 banks in the next 21 CPs. Each bank is referenced twice during this time, once for a read and once for a write. A fetch request follows immediately after the exchange reference is complete and then referencing from the four memory ports of the other CPU is enabled.

Conflict resolution

During each clock period, references to the memory ports in the system are examined for memory access conflicts. If a conflict occurs for a reference, the reference is held and no further referencing from that port is allowed until the conflict is resolved.

Three types of memory access conflicts can occur: Bank Busy, Simultaneous Bank, and Section.

Bank Busy conflict - The Bank Busy conflict is caused by any port within or between CPUs requesting a bank currently in a reference cycle. Resolution of this conflict occurs when the bank cycle is complete. Hold reference because of a Bank Busy conflict is 1, 2, or 3 CPs.

Simultaneous Bank conflict - The Simultaneous Bank conflict is caused by two or more ports in different CPUs requesting the same bank. Resolution of this conflict is based on a priority (see Inter-CPU priority). Hold reference is 1 CP because of a Simultaneous Bank conflict. A Bank Busy conflict always follows a Simultaneous Bank conflict.

Section conflict - The Section conflict is caused by two or more ports in the same CPU requesting any bank in the same section. Resolution of this conflict is based on a priority, the Bank Busy conflict, and Simultaneous Bank conflict. The highest priority port with no Bank Busy conflict and no Simultaneous Bank conflict is allowed to proceed, all other ports involved in this conflict hold (see Intra-CPU priority). Hold reference is 1 CP because of a Section conflict.

Memory access priorities

The following priorities are used to resolve memory access conflicts.

- Intra-CPU priority: the priority between Ports A, B, and C is determined by the following conditions:

  - Any port with an odd increment always has a higher priority than a port with an even increment regardless of their issued sequence.

  - Among all ports with the same type of increment (odd or even), the relative time of issue determines the priority, with the first issued having the highest priority.

- Inter-CPU priority: every 4 clock periods the priority between CPUs changes.
- I/O priority: the I/O ports are always lowest priority, within or between CPUs.

16-BANK PHASING

The effect of 16-bank phasing on instruction fetches is a predictable increase of 2 CPs for filling an instruction buffer. Otherwise, the amount of performance degradation for 16 banks compared with 32 banks is not readily predictable since it largely results from an increase of memory conflicts.

For maintenance purposes, a 32-bank system can be modified to operate with only 16 banks using either the lower or upper half of memory. Maintenance is accomplished by setting the bank select switch to the lower or upper banks.

MEMORY ERROR CORRECTION

A single error correction/double error detection (SECDED) network is used between a CPU and memory. SECDED assures that data written into memory can be returned to the CPU with consistent precision (figure 2-4).

If a single 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 error is detected but not corrected. In either case, the CPU can be interrupted depending on interrupt options selected to allow processing of the error. For three or more bits in error, results are ambiguous.

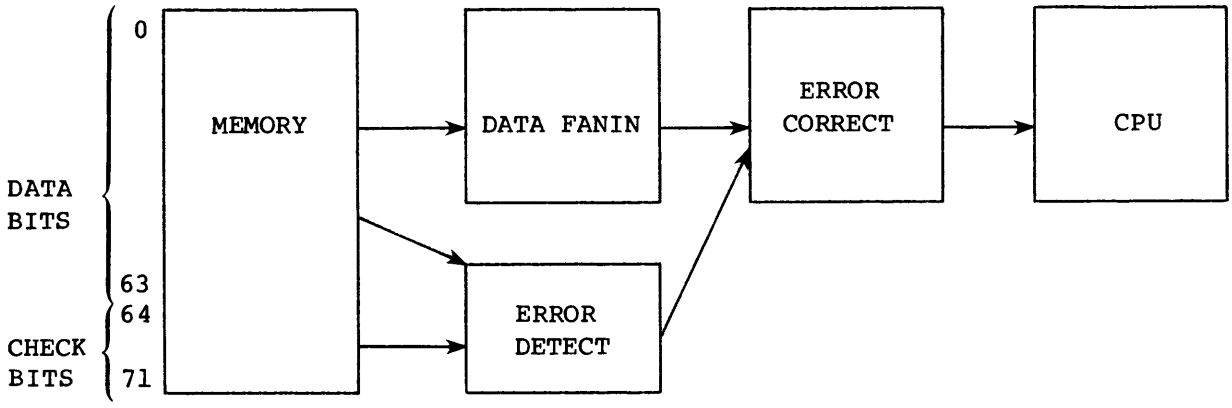


Figure 2-4. Memory data path with SECDED

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 8 check bits are each generated as even parity bits for a specific group of data bits. Figure 2-5 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 0 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 8 check bits and the data word are stored in memory at the same location. When read from memory, the same 64-bit matrix of figure 2-5 is used to generate a new set of check bits, which are compared with the old check bits. The resulting 8 comparison bits are called syndrome†† bits (S bits). The states of these S bits are all symptoms of any error that occurred (1 = no compare). If all syndrome bits are 0, no memory error is assumed.

Any change of state of a single bit in memory causes an odd number of syndrome bits to be set to 1. A double error (an error in two bits) appears as an even number of syndrome bits set to 1.

The matrix is designed so that:

- If all syndrome bits are 0, no error is assumed.
- If only 1 syndrome bit is 1, the associated check bit is in error.
- If more than 1 syndrome bit is 1 and the parity of all syndrome bits S0 through S7 is even, then a double error (or an even number of bit errors) occurred within the data bits or check bits.
- If more than 1 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.
- If 3 or more memory bits are in error, the parity of all syndrome bits is odd and results are ambiguous.

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

†† Syndrome: Any set of characteristics regarded as identifying a certain type, condition, etc. Websters New World Dictionary.

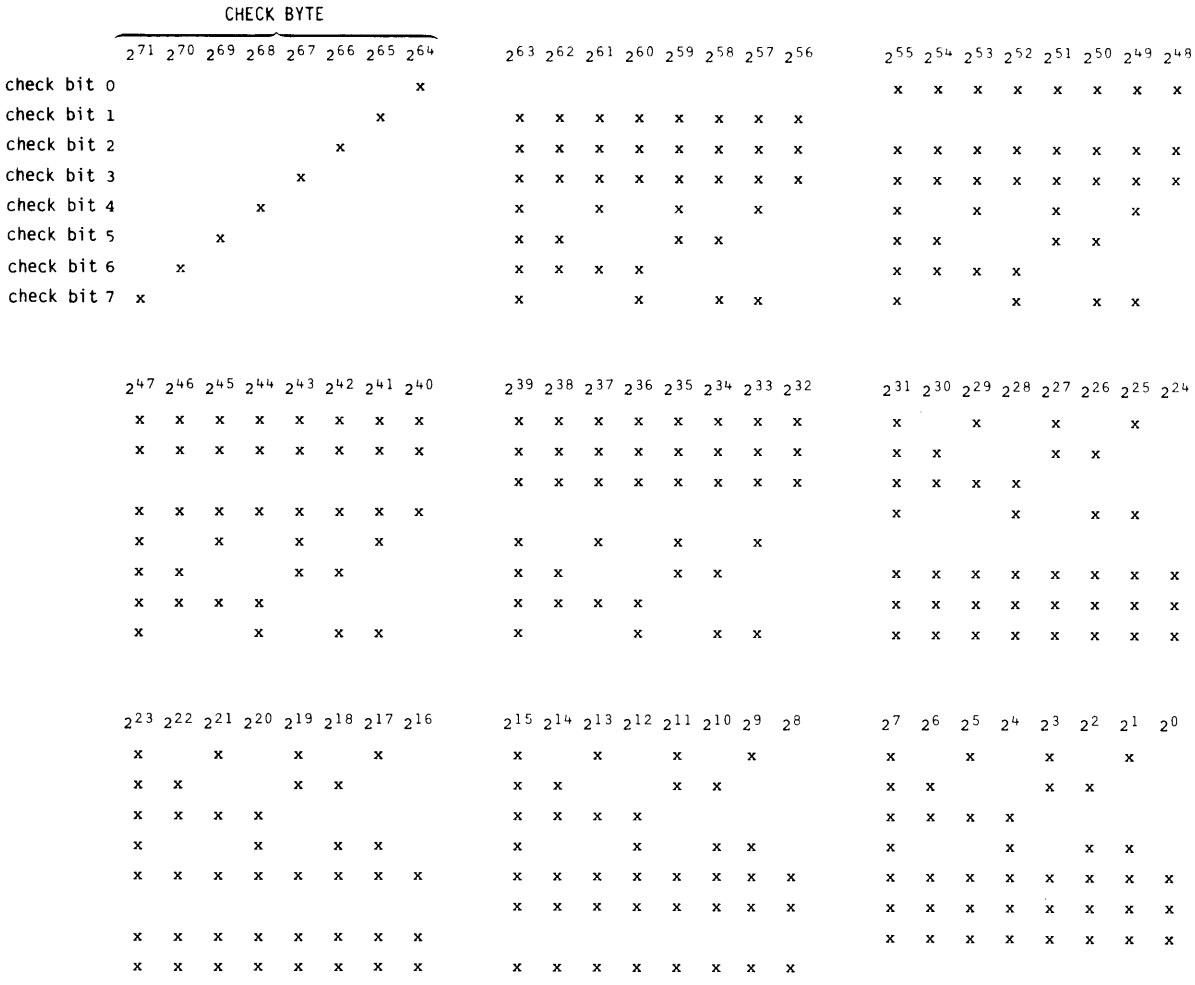


Figure 2-5. Error correction matrix

INTER-CPU COMMUNICATION SECTION

The inter-CPU communication section of the CRAY X-MP mainframe contains special hardware for communication between the two CPUs, for control, and for a real-time clock. The Real-time Clock (RTC), Shared Address (SB), Shared Scalar (ST), and Semaphore (SM) registers are shared by the CPUs. These registers with their sources and destinations are shown in figure 2-6 and described in the following paragraphs.

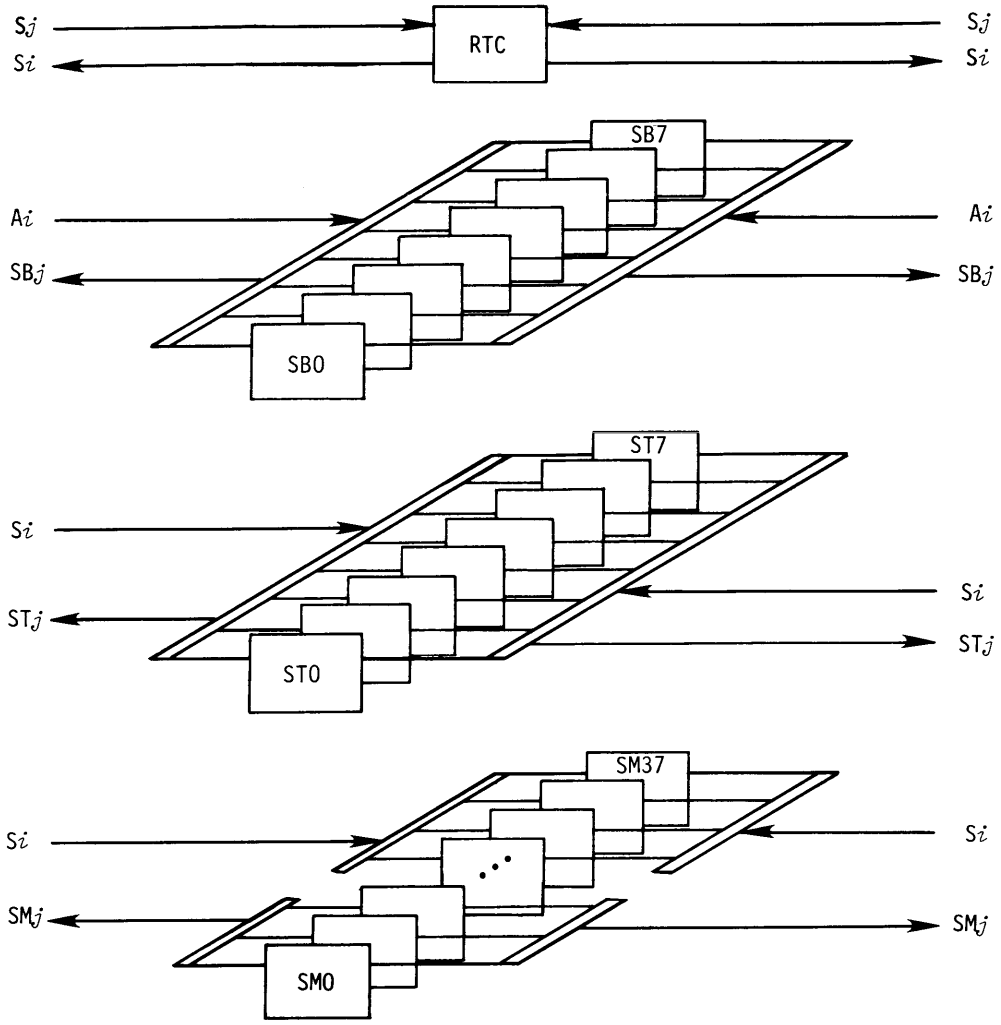


Figure 2-6. Shared registers

REAL-TIME CLOCK

The CRAY X-MP mainframe contains one Real-time Clock (RTC) register shared by the two CPUs. Programs can be timed precisely by using the clock period (CP) counter. This counter is 64 bits wide and advances one count each CP of 9.5 nanoseconds. Since the clock advances synchronously with program execution, it can be used to time the program to an exact number of CPs. However, in such an application, the counting can contain counts from other tasks if an interrupt occurs before the end time is read.

Instructions used with the Real-time Clock (RTC) register are:

0014j0	RT	Sj	Enter the RTC register with (Sj)
072i00	Si	RT	Transmit (RTC) to Si

The CP counter can be read by a program using instruction 072 and can be reset only by instruction 0014j0. Loading or reading the CP counter can occur from both CPUs at the same time. If both CPUs are in monitor mode, the software should ensure that only one CPU enters a value into this register.

INTER-CPU COMMUNICATION AND CONTROL

Three identical sets of shared registers in the CRAY X-MP are used for communication and control between the two CPUs. Each set contains eight 24-bit Shared Address (SB) registers, eight 64-bit Shared Scalar (ST) registers and thirty-two 1-bit Semaphore (SM) registers.

Each CPU's Cluster Number (CLN) register determines which set of shared registers is accessed by a CPU (clustering). The CLN register is loaded from the exchange package or if the CPU is in monitor mode, via instruction 0014j3. The CLN register can contain one of four different values. Values 1, 2, or 3 allow the CPU to access one of the three sets of shared registers. Value 0 prevents any access to shared registers by the CPU. If the value is 0, instructions regarding the shared registers become no-ops, except for the instructions returning values to Ai or Si, which return a 0 value. If the CLN registers in both CPUs are set to the same value (1, 2, or 3), then the two CPUs share a common set of SB, ST, and SM registers.

Shared Address and Shared Scalar registers

The Shared Address (SB) and Shared Scalar (ST) registers are used for passing address and scalar information from one CPU to another. No hardware reservations are made on these registers. Any necessary reservations to restrict access to these registers must be handled in the software through use of the Semaphore (SM) registers or by shared memory design. The single hardware restriction on access to the SB and ST registers is that only one read (occurs at issue) or one write (occurs 3 CPs after issue) operation can occur in a CP.

The instructions used with the SB and ST registers are:

026ij7	Ai	SBj	Transmit (SBj) to Ai
027ij7	SBj	Ai	Transmit (Ai) to SBj
072ij3	Si	STj	Transmit (STj) to Si
073ij3	STj	Si	Transmit (Si) to STj

Access conflicts to Shared Address (SB) and Shared Scalar (ST) registers occur under the conditions shown in table 2-1 regardless of clustering. For example, if a read instruction for CPU 0 and a read instruction for CPU 1 enter CIP simultaneously, a conflict occurs and CPU 1 holds issue for one CP.

Table 2-1. Access conflicts to shared registers

CPU 0 SB or ST register operation	CPU 1 SB or ST register operation	Hold issue 1 CP
READ (first CP in CIP)	READ (first CP in CIP)	CPU 1
READ (not first CP in CIP)	READ (first CP in CIP)	CPU 1
READ (first CP in CIP)	READ (not first CP in CIP)	CPU 0
READ (not first CP in CIP)	READ (not first CP in CIP)	CPU 0
WRITE (first CP in CIP)	WRITE (first CP in CIP)	CPU 1
WRITE (not first CP in CIP)	WRITE (first CP in CIP)	CPU 1
WRITE (first CP in CIP)	WRITE (not first CP in CIP)	CPU 0
WRITE (not first CP in CIP)	WRITE (not first CP in CIP)	CPU 0
READ (Write issued 3 CPs before)		CPU 0
READ	(Write issued 3 CPs before)	CPU 0
	READ (Write issued 3 CPs before)	CPU 1
(Write issued 3 CPs before)	READ	CPU 1

Semaphore registers

The Semaphore (SM) registers are used for control between the two CPUs. No hardware reservations are made on these registers. Loading or reading the SM registers or setting or clearing a particular SM register can occur at any time from either or both CPUs.

The test and set instruction (0034*jk*) is the only operation on the SM registers including a hardware interlock. This interlock prevents a simultaneous test and set operation on the same SM register from both

CPUs. In this case, CPU 1 holds issue and CPU 0 proceeds. The test and set instruction first tests the value of the selected SM register. If the value is 0, the instruction issues and sets that SM register to a 1. If the value is 1, the instruction holds issue until the value is 0.

When both CPUs in a cluster are holding issue on a test and set instruction, a deadlock interrupt can occur. If the CLN registers in both CPUs are equal and not 0, both CPUs belong to the same cluster and both CPUs must be holding issue on a test and set instruction to cause a deadlock interrupt. In this case, both CPUs receive a deadlock interrupt. If the CLN registers in both CPUs are not equal, the two CPUs are in different clusters. If one CPU holds issue on a test and set instruction, that CPU receives a deadlock interrupt. No deadlock interrupt can occur in cluster 0 (CLN = 0).

When an interrupt occurs, normally the instructions already in the NIP and CIP registers are allowed to issue before the exchange sequence starts. If a test and set instruction is holding in the CIP register and an interrupt occurs, a special exchange start-up sequence is initiated. In this case the instruction in the NIP register and the test and set instruction in the CIP register are discarded and the program counter (P) register is adjusted to point to the discarded test and set instruction. The Waiting on Semaphore (WS) flag in the exchange package sets, indicating a test and set instruction was holding in the CIP register when the interrupt occurred. The exchange sequence is then started.

Instructions used with the SM registers are:

0034*jk*	SM*jk*	1,TS	Test and set, SM*jk*
0036*jk*	SM*jk*	0	Clear SM*jk*
0037*jk*	SM*jk*	1	Set SM*jk*
072*i*02	S*i*	SM	Transmit (SM) to S*i*
073*i*02	SM	S*i*	Transmit (S*i*) to SM

CPU INPUT/OUTPUT SECTION

The Input/Output section of the CRAY X-MP mainframe is shared by the two Central Processing Units (CPUs). The CRAY X-MP supports three channel types identified by their maximum transfer rates of 1250 Mbytes per second, 100 Mbytes per second, and 6 Mbytes per second.

One 1250 Mbytes per second channel transfers data between the Central Memory and the Solid-state Storage Device (SSD). This channel is 128 bits wide and uses 16 check bits in each direction. A maximum transfer rate of over 10 gigabits per second is possible on the 1250 Mbytes per second channel. The channel is two parallel 64-bit channels each with SECDED; therefore, under certain circumstances the full-width channel can correct double errors.

Two 100 Mbytes per second channels transfer data between Central Memory and the I/O Subsystem. A 100 Mbytes per second channel is 64 bits wide and uses 8 check bits in each direction. Data words are transferred in blocks of 16 under control of Data Ready and Data Transmit control signals. Each 100 Mbytes per second channel has a maximum transfer rate of approximately 850 Mbits per second.

I/O Subsystem communication with the CPUs is over four control channels, each with a maximum transfer rate of 6 Mbytes per second. Each 6 Mbytes per second channel is 16 bits wide.

There are two I/O ports, one from each CPU. The channels are hardwired into a port with two 6 Mbytes per second channel pairs and one or two 100 Mbytes per second channel pairs per port. Each port can transfer data at a rate of one word per clock period (CP). For the 100 Mbytes per second channels, each time a buffer makes a reference it holds the port to completion, usually 16 words.

All I/O (including 100 Mbytes per second channels) uses the I/O ports to memory. Access to these ports is controlled by a scanner. All CPU memory ports (Ports A, B, and C) have higher priority than the I/O ports.

Channel features of the input/output section are summarized below and described in the remainder of this section.

- One 1250 Mbytes per second channel, maximum transfer rate
  - 128 data bits and 16 check bits in each direction

- Two 100 Mbytes per second channels, maximum transfer rate per channel
  - 64 data bits, 3 control bits, and 8 check bits in each direction

- Up to four I/O channels, 6 Mbytes per second maximum transfer rate per channel
  - Shared control from the two CPUs
  - 16 data bits, 3 control bits, and 4 parity bits in each direction
  - Lost data detection

- Channels are divided into four groups, each group contains either two input or two output channels

- Channel groups are served equally by memory (each group is scanned every 4 CPs)

- Channel priority resolved within channel groups

DATA TRANSFER FOR SOLID-STATE STORAGE DEVICE

Data is transferred directly between the Solid-state Storage Device (SSD) and the CRAY X-MP mainframe using the 1250 Mbytes per second channel. The 1250 Mbytes per second channel is 128 bits wide and is programmed through software. Port 3 of the SSD connects with the CRAY X-MP. Programming details for the SSD are described in the Solid-state Storage Device (SSD) Reference Manual, CRI publication HR-0031.

DATA TRANSFER FOR I/O SUBSYSTEM

A 100 Mbytes per second channel transfers data between Central Memory of the CRAY X-MP and the Buffer I/O Processor (BIOP) of the I/O Subsystem. A second 100 Mbytes per second channel transfers data between Central Memory and a Disk I/O Processor (DIOP) or Auxiliary I/O Processor (XIOP). (Software does not currently support data transfer using the 100 Mbytes per second channel to an XIOP.) Each channel is 64 bits wide and handles data at approximately 100 Mbytes per second. Each channel uses an additional 8 check bits for single error correction/double error detection (SECDED), as is used in Central Memory.

The CPU side of a 100 Mbytes per second channel uses a pair of 16-word buffers to stream the data out of Central Memory and another pair to stream data into Central Memory. On output, as one buffer block is being sent to the I/O Processor (IOP), the other buffer is filling from Central Memory. Similarly, on input, one buffer block is filling from an IOP while the other is transmitting to Central Memory.

At the IOP side of a 100 Mbytes per second channel, data passing into Local Memory (an I/O Processor's memory) is double-buffered and disassembled into 16-bit parcels. The channel side passing data from Local Memory simply assembles 16-bit parcels into 64-bit words for transmission to a CPU.

An I/O Processor controls a 100 Mbytes per second channel linking it with Central Memory. The IOP initiates all data transfers on the channel and performs all error processing required for the channel. There are no CPU instructions for the 100 Mbytes per second channel. Programming details for the 100 Mbytes per second channel are contained in the CRAY I/O Subsystem Reference Manual, publication HR-0030.

6 MBYTES PER SECOND CHANNELS

Standard control channels for the CRAY X-MP are 6 Mbytes per second channels. Each 6 Mbytes per second channel has 16-bit asynchronous control logic used for front-end interfaces. The instructions used with 6 Mbytes per second channels follow.

0010*jk*	CA,A*j* A*k*	Set the Current Address (CA) register for the channel indicated by (A*j*) to (A*k*) and activate the channel
0011*jk*	CL,A*j* A*k*	Set the Limit Address (CL) register for the channel indicated by (A*j*) to (A*k*)
0012*jk*	CI,A*j*	Clear the interrupt flag and error flag for the channel indicated by (A*j*): Output channel *k*=0; clear MC, *k*=1; set MC. Input channel *k*=0; no operation, *k*=1; clear held ready.
033*i*00	A*i* CI	Transmit channel number to A*i*
033*ij*0	A*i* CA,A*j*	Transmit address of channel (A*j*) to A*i*
033*ij*1	A*i* CE,A*j*	Transmit error flag of channel (A*j*) to A*i*

MULTI-CPU PROGRAMMING

The four 6 Mbytes per second I/O channels can operate from either CPU, and either CPU can issue instructions to any of the channels. There is no hardware interlock between the two CPUs; therefore, software must ensure that only one CPU is servicing I/O at a time, while in monitor mode. Instruction 033 is independent in nature and can be issued without an interlock.

The following conditions must be met for an I/O interrupt to occur.

- Neither CPU is waiting for an exchange.
- Neither CPU is in monitor mode.
- An interrupt is present.

Normally, the interrupt from a 6 Mbytes per second channel is directed toward the CPU that last issued a clear interrupt instruction (0012) to that channel. However, because an I/O interrupt occurs in only one CPU at a time, the following conditions (in priority order) determine the CPU toward which the interrupt is directed. Once in monitor mode, a CPU should service all I/O interrupts.

1. All I/O interrupts are directed toward a CPU that has the Select External Interrupt Mode set.

2. If neither CPU has selected external interrupts, then interrupts are directed toward a CPU holding issue on a test and set instruction.

3. If neither conditions 1 nor 2 exist or if they exist in both CPUs, the interrupt is directed to the CPU that last issued a clear interrupt instruction to that channel.

6 MBYTES PER SECOND CHANNEL OPERATION

Each input or each output channel directly accesses the Central 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 Central Memory words into 16-bit parcels or 16-bit parcels into 64-bit Central Memory words. Four parcels make up one Central Memory word with bits of the parcels assigned to memory bit positions as shown in table 2-2. In both input and output operations, parcel 0 is always transferred first.

Each input or output channel has a data channel (4 parity bits, 16 data bits, and 3 control lines), a 64-bit assembly or disassembly register, a channel Current Address (CA) register, and a channel Limit Address (CL) register.

Three control signals (Ready, Resume, and Disconnect) coordinate the transfer of parcels over the channels. In addition to the three control signals, the output channel of a pair has a Master Clear line. Appendix B describes the signal sequence of a 6 Mbytes per second channel.

I/O interrupts can be caused by the following:

- On all output channels, if (CA) becomes equal to (CL), then the resume for the last parcel transmitted sets interrupt.

- External device disconnect is received on any input channel and channel is active

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

The number of the channel causing an interrupt can be determined by using instruction 033, which reads into 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 when an interrupt from the next highest

priority channel, if present, is sensed. All interrupts are available via instruction 033 to either CPU. Channel numbers for 6 Mbytes per second channels range from 10_8 through 17_8 (10/11, 12/13, 14/15, and 16/17 - even for output/odd for input).

Table 2-2. 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 X-MP 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

INPUT CHANNEL PROGRAMMING

To start an input operation, the CPU program:

1. Sets the channel limit address to the last word address + 1 (LWA+1). (See figure 2-7.)

2. Sets the channel current address to the first word address (FWA).

Setting the current address causes the Channel Active flag to set. 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 CA register. When the word is accepted by memory, the current address is advanced by 1.

An external transmitting device sends a Disconnect signal to indicate end of a transfer. When the Disconnect signal is received, the Channel Interrupt flag sets and a test is performed to check for a partially assembled word. If the partial word is found, the valid portion of the word is stored in memory and the unreceived, low-order parcels are stored as zeros.

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

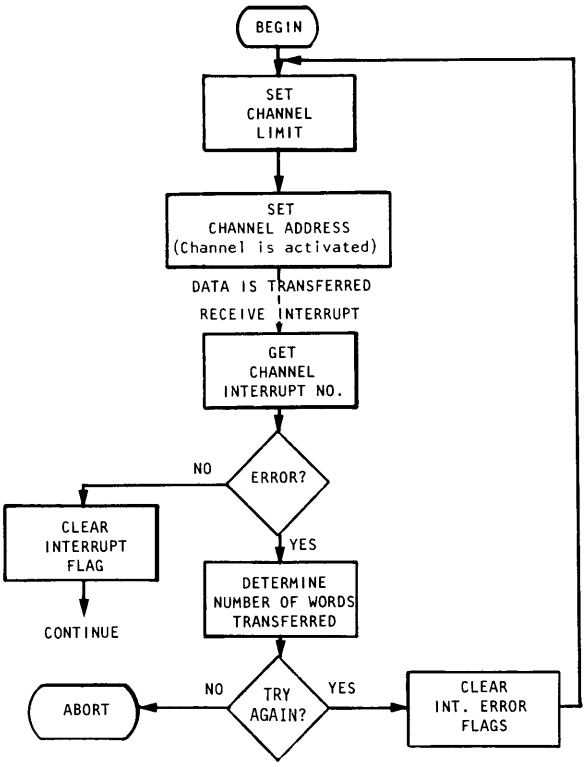


Figure 2-7. Basic I/O program flowchart

INPUT CHANNEL ERROR CONDITIONS

Input channel error conditions can occur at a parcel level (parity error) or channel level (unexpected Ready signal). When a parcel in error occurs, the Parity Fault flag sets immediately. The Parity Fault flag does not generate an interrupt, it is saved and sets the error flag when a disconnect occurs. Therefore, the program should check the state of the error flag when an interrupt is honored. All parcels stored after the error are zeroed.

If a Ready signal is received when the channel is not active (unexpected Ready signal), the Ready condition is held until the channel is activated. At this time a Resume signal is sent. No error flag is set and no interrupt request is generated. Since the Ready condition is held when the channel is inactive, it is sometimes advantageous to be able to clear this Ready signal before setting up the channel, especially on a deadstart or a resynchronization of the channel after an error. The Ready signal can be cleared by using instruction 0012j1 to input channel (Ai), clearing any Ready signal being held before issue of instruction 0012j1.

OUTPUT CHANNEL PROGRAMMING

To start an output operation, the CPU program:

1. Sets the channel limit address to the last word address + 1 (LWA+1).

2. Sets 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 CA register. When the word is received from memory, the channel advances the current address by 1 and starts the data transfer.

After each word is read from memory and the current address is advanced, the limit test is made, comparing the contents of the CA register and the CL register. If they are equal, the operation is complete as soon as the last parcel transfer is finished.

The interrupt flag also sets if an error is detected. The only error that an output channel detects is a Resume signal received when the channel is inactive. No external response is generated.

PROGRAMMED MASTER CLEAR TO EXTERNAL DEVICE

The CRAY X-MP can send a Master Clear signal to an external device through the output channel. The external Master Clear sequence is as follows.

1. 0012*jk* Clears input channel to ensure external activity on the channel pair has stopped.

2. 0012*j*1 Clears output channel to ensure CPU activity on the channel pair has stopped. Set Master Clear.

3. Delay 1 Device dependent; determines the duration of the Master Clear signal.

4. 0012*j*0 Clears the output channel. This turns off the Master Clear signal.

5. Delay 2 Device dependent; allows time for initialization activities in the attached device to complete.

For Cray Research, Inc., front-end interfaces, delays 1 and 2 should each be a minimum of 80 CPs.

MEMORY ACCESS

Each of the four channel groups shown below is assigned a time slot (figure 2-8) that is scanned once every 4 CPs for a memory request. The lowest numbered channel in the group has the highest priority. During the next 3 CPs, the scanner allows requests from the other three channel groups. Therefore, it is possible to have an I/O memory request every CP. The scanner stops for all memory conflicts caused by an I/O reference and also stops for a block (100 Mbytes per second channel) reference while a buffer is referencing, maximum 16 words (figure 2-9).

The 6 Mbytes per second channels are numbered 10_8 through 17_8 and the 100 Mbytes per second channels are numbered 0 to 7. The channels are grouped as follows:

	CPU 0	CPU 1
Group 0 input channels	0,10	2,12
Group 1 output channels	1,11	3,13
Group 2 input channels	4,14	6,16
Group 3 output channels	5,15	7,17

I/O LOCKOUT

An I/O memory request can be locked out by an exchange sequence or instruction fetch sequence.

MEMORY BANK CONFLICTS

Memory bank conflicts are tested for CPU scalar, vector, and I/O memory references. When a exchange sequence or instruction fetch sequence is in progress, all other memory references are locked out.

Each memory bank can accept a new request every 4 CPs. To test for a memory bank conflict, the 5 low-order bits† of the memory address are checked against Bank Busy conflicts and other memory references. The bank is busy for 4 CPs on a reference.

† 4 bits for 16-bank phasing; refer to subsection on Central Memory.

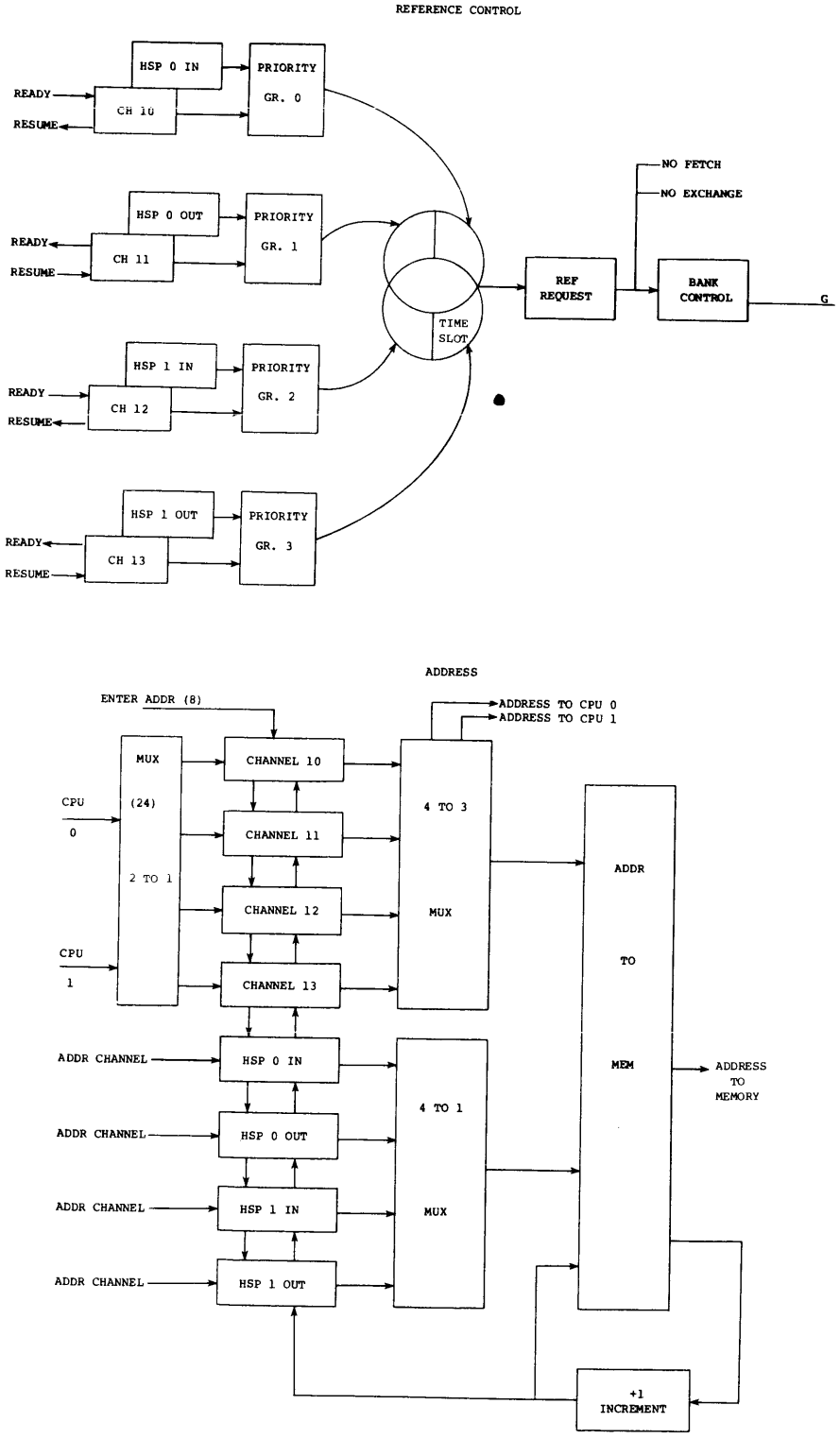


Figure 2-8. Channel I/O control (shown for one processor)

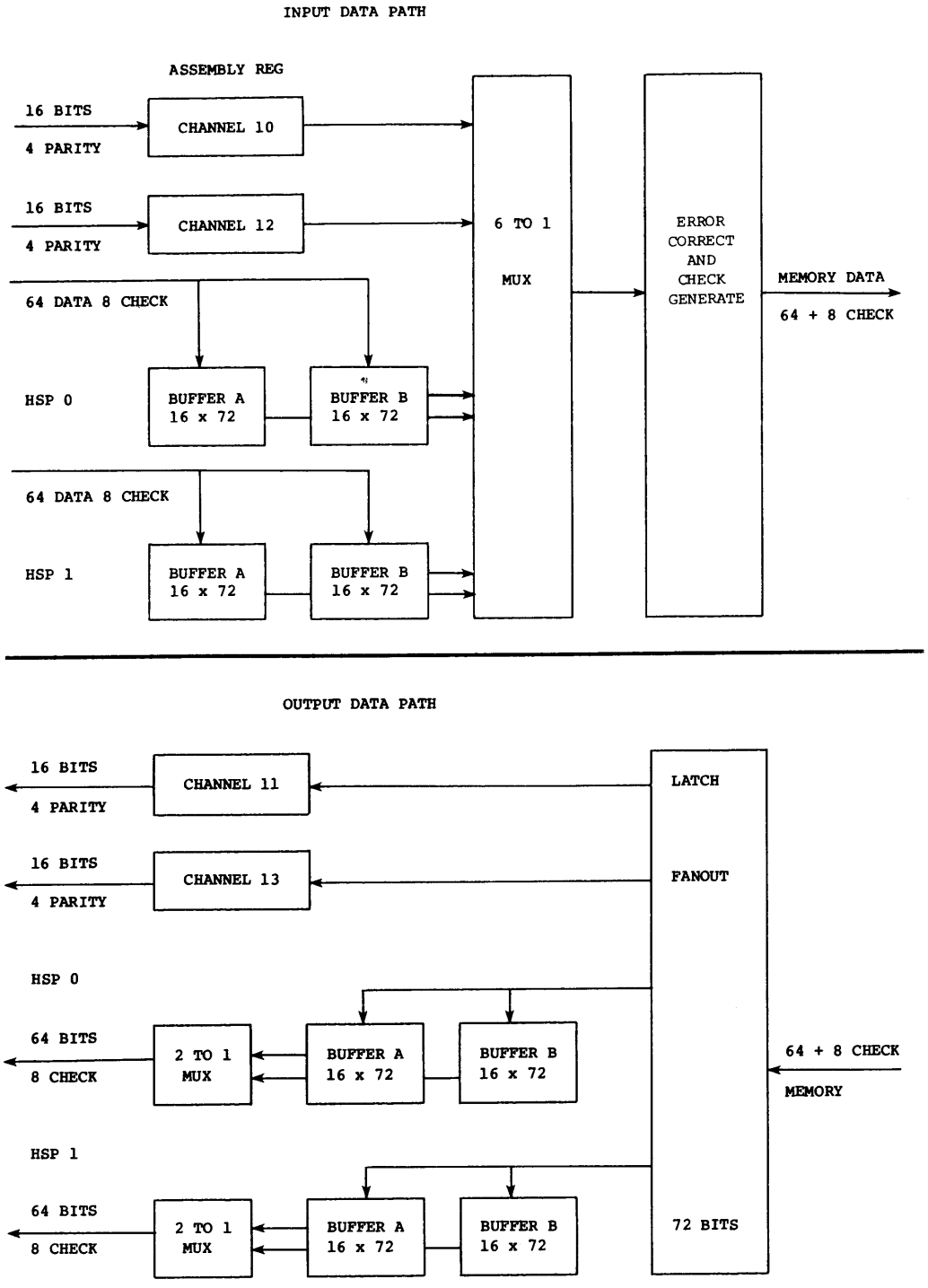


Figure 2-9. Input/output data paths

I/O MEMORY CONFLICTS

Before testing for a memory bank conflict, a check is made to ensure no exchange sequence or instruction fetch sequence is in progress. If either of these conditions exists, the I/O request is held. The 5 low-order address bits[†] of an I/O reference are tested against Bank Busy conflicts and other memory references. If a bank being referenced is busy, the reference is held and the scanner is stopped.

I/O MEMORY REQUEST CONDITIONS

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

- I/O request
- Bank not busy
- No simultaneous conflicts with other memory ports
- No fetch request
- No exchange sequence

I/O MEMORY ADDRESSING

All I/O memory references are absolute. The CA and CL registers are 22 bits, allowing I/O access to all of memory. Setting of the CA and CL registers is limited to monitor mode. I/O memory reference addresses are not checked for range errors.

---

† 4 bits for 16-bank phasing; refer to subsection on Central Memory.

CPU CONTROL SECTION 3

INTRODUCTION

Each CPU contains an identical, independent control section containing registers and instruction buffers for instruction issue and control. A control section uses an exchange mechanism for switching instruction execution from program to program. These registers and buffers and the exchange mechanism are described in this section. Memory field protection, programmable clock, and deadstart sequence are also described.

INSTRUCTION ISSUE AND CONTROL

The registers and instruction buffers involved with instruction issue and control are described in the following paragraphs. Figure 3-1 illustrates the general flow of instruction parcels through the registers and buffers.

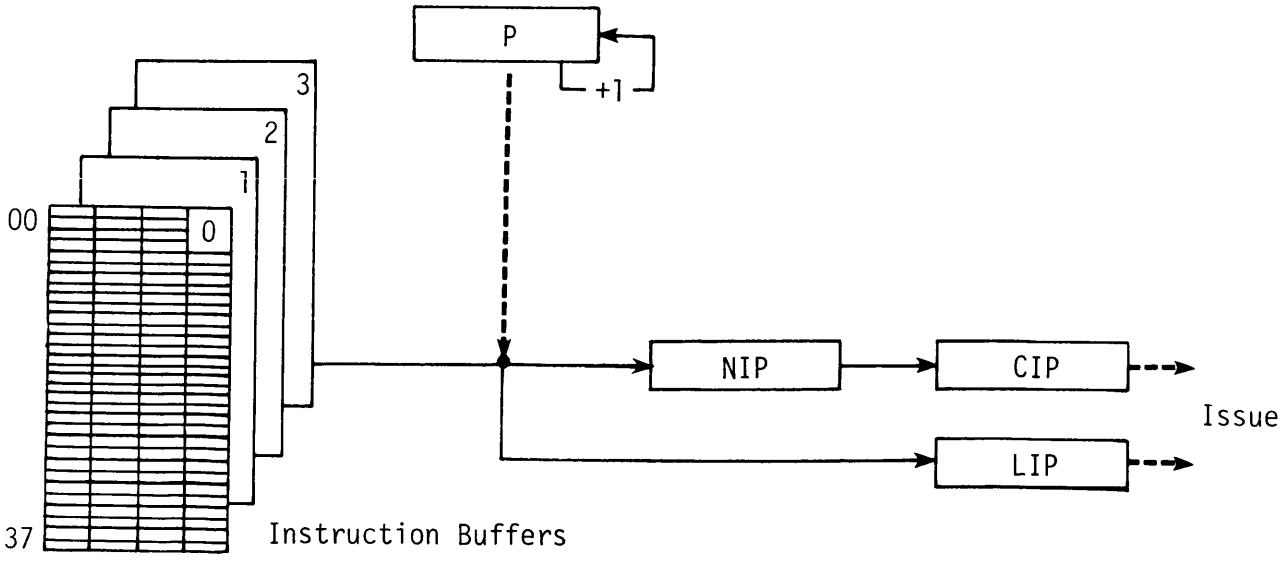


Figure 3-1. Instruction issue and control elements

PROGRAM ADDRESS REGISTER

The 24-bit Program Address (P) register indicates the next parcel of program code to enter the Next Instruction Parcel (NIP) register. The high-order 22 bits of the P register indicate the word address for the program word in memory. The low-order 2 bits indicate the parcel within the word. Except on a branch instruction when the branch is taken or on an exchange, the contents of the P register are advanced 1 when an instruction parcel enters the NIP register.

New data enters the P register on an instruction branch or on an exchange sequence. (The exchange sequence is described under Exchange Mechanism later in this section.) The contents of P are 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. The value stored in P might not be accurate during the deadstart sequence.

NEXT INSTRUCTION PARCEL REGISTER

The 16-bit Next Instruction Parcel (NIP) register holds a parcel of program code before it enters the Current Instruction Parcel (CIP) register.

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

CURRENT INSTRUCTION PARCEL REGISTER

The 16-bit Current Instruction Parcel (CIP) register holds the instruction waiting to issue. The term *issue* indicates the transition of an instruction in CIP to its execution phase. If an instruction is a 2-parcel instruction, the CIP register holds the first parcel of the instruction and the Lower Instruction Parcel (LIP) register holds the second parcel. Issue of an instruction in CIP can be delayed until conflicting operations have been completed. Data arrives at the CIP register from the NIP register. Indicators making up the instruction are distributed to all modules having mode selection requirements when the instruction issues.

The control flags associated with the CIP register are master cleared; the register itself is not. An undetermined instruction can issue during the master clear sequence.

LOWER INSTRUCTION PARCEL REGISTER

The 16-bit Lower Instruction Parcel (LIP) register holds the second parcel of a 2-parcel instruction at the time the first parcel of the 2-parcel instruction is in the CIP register.

INSTRUCTION BUFFERS

A CPU has four instruction buffers, each can hold 128 consecutive 16-bit instruction parcels (figure 3-2). Instruction parcels are held in the buffers before being delivered to the NIP or LIP registers.

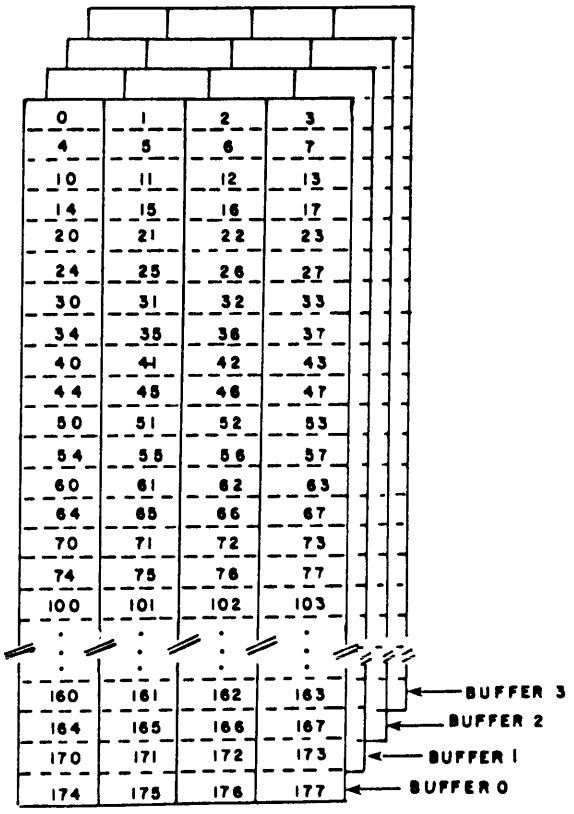


Figure 3-2. Instruction buffers

The beginning instruction parcel in a buffer always has a word address that is a multiple of 40_8 (a parcel address that is a multiple of 200_8) allowing the entire range of addresses for instructions in a buffer to be defined by the high-order 17 bits of the parcel address. Each buffer has a 17-bit beginning address register containing this value.

The beginning address registers are scanned each clock period (CP). If the high-order 17 bits of the P register match one of the beginning

addresses, an in-buffer condition exists and the proper instruction parcel is selected from that instruction buffer. An instruction parcel to be executed normally is sent to the NIP. However, the second parcel of a 2-parcel instruction is blocked from entering the NIP register and is sent to the LIP register instead. The second parcel of the 2-parcel instruction becomes available when the first parcel issues from the CIP register. At the same time, an all-zero parcel is entered into the NIP register.

On an in-buffer condition, if the instruction is in a different buffer than the previous instruction, a change of buffers occurs requiring a 2-CP delay of the instruction reaching the NIP register.

An out-of-buffer condition exists when the high-order 17 bits of the P register do not match any instruction buffer beginning address. When this condition occurs, instructions must be loaded from memory into one of the instruction buffers before execution can continue. A 2-bit counter determines the instruction buffer receiving the instructions. Each out-of-buffer condition causes the counter to be incremented by 1 so that the buffers are selected in rotation.

Buffers are loaded from memory at the rate of eight words per CP, fully occupying memory. The first group of 32 parcels delivered to the buffer always contains the next instruction required for execution. For this reason, the branch out-of-buffer time is 16 CPs for 32-bank memories and 18 CPs for 16-bank memories providing memory is not busy (if busy, the branch fetch is delayed until the busy is resolved). Once the fetch proceeds, the remaining groups arrive at a rate of 32 parcels per CP and circularly fill the buffer.

An instruction buffer is loaded with one word of instructions from each of the 32 memory banks or two words from each of the 16 banks. The first four instruction parcels residing in an instruction buffer are always from bank 0. An exchange sequence voids the instruction buffers, preventing a match with the P register and causing the buffers to be loaded as needed.

Forward and backward branching are possible within buffers. Branching does not cause reloading of an instruction buffer if the address of the instruction 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 before issue and after (until the buffer is reloaded), self-modifying code should not be used. Also, because there is independent data and instruction memory protection, self-modifying code may be impossible. As long as the address of the unmodified instruction is in an instruction buffer, the modified instruction in memory is not loaded into an instruction buffer.

Although optimizing code segment lengths for instruction buffers is not a prime consideration when programming a CPU, the number and size of the buffers and the capability for forward and backward branching can be used to good advantage. Large loops containing up to 512 consecutive

instruction parcels can be maintained in the four buffers. An alternative is for a main program sequence in one or two of the buffers to make repeated calls to short subroutines maintained in the other buffers. The program and subroutines remain undisturbed in the buffers as long as no out-of-buffer condition or exchange causes reloading of a buffer.

EXCHANGE MECHANISM

A CPU uses an exchange mechanism for switching instruction execution from program to program. This exchange mechanism involves the use of blocks of program parameters known as Exchange Packages and a CPU operation referred to as an exchange sequence. For the convenience of Cray Assembler Language (CAL) programmers, an alternate bit position representation is used when discussing the Exchange Package. The bits are numbered from left to right with bit 0 assigned to the 2^{63} bit position.

EXCHANGE PACKAGE

The Exchange Package (figure 3-3) is a 16-word block of data in memory associated with a particular computer program. The Exchange Package contains the basic parameters necessary to provide continuity from one execution interval for the program to the next. These parameters are listed below and are described on the following pages.

Field	Word	Bits
Processor number (PN)	0	1
Error type (E)	0	2-3
Syndrome bits (S)	0	4-11
Program Address register (P)	0	16-39
Read mode (R)	1	0-1
Read address (CSB)	1	2-6 (CS); 7-11 (B)
Instruction Base Address (IBA)	1	18-34
Instruction Limit Address (ILA)	2	18-34
Mode register (M)	1-2	35-39
Vector not used (VNU)	2	0
Flag register (F)	3	14-15; 31-39
Exchange Address register (XA)	3	16-23
Vector Length register (VL)	3	24-30
Data Base Address (DBA)	4	18-34
Program State (PS)	4	35
Cluster Number (CLN)	4	38-39
Data Limit Address (DLA)	5	18-34
Current contents of the eight A registers	0-7	40-63
Current contents of the eight S registers	8-15	0-63

The Exchange Package contents are arranged in a 16-word block. The exchange sequence swaps data from memory to the operating registers and back to memory. This sequence exchanges data in an active Exchange Package residing in the operating registers with an inactive Exchange Package in memory. The Exchange Address (XA) register address of the active Exchange Package specifies the memory address to be used for the swap. Data is exchanged and a new program execution interval is initiated by the exchange sequence.

The contents of the B, T, V, VM, SB, ST, and SM registers are not swapped in the exchange sequence. Data in these registers must be stored and replaced as required by specific coding in the program supervising the object program execution or by any program that needs this data. (See section 4 for descriptions of the operating registers and the VL register.)

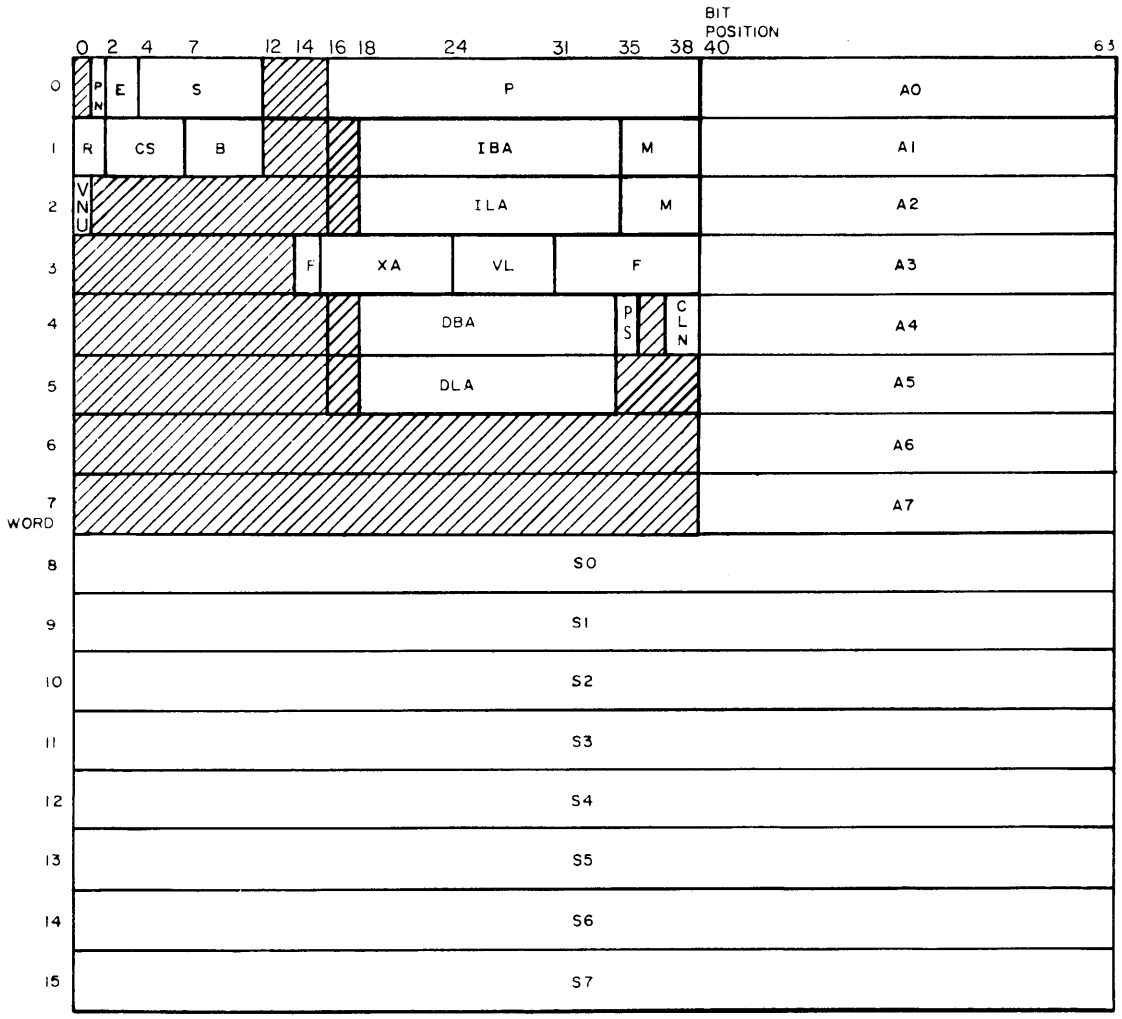


Figure 3-3. Exchange package

Processor Number

The content of the 1-bit processor number (PN) position in the Exchange Package indicates in which CPU (0 or 1) the Exchange Package executed. This value is not read into the CPU; it is a constant inserted only into a package being stored.

Memory error data

Bit 36 (interrupt on correctable memory error bit) and bit 38 (interrupt on uncorrectable memory error bit) in the M (mode) register determine if memory error data is included in the Exchange Package. Error data, consisting of four fields of information, appears in the Exchange Package if bit 36 is set and correctable memory error is encountered or if bit 38 is set and an uncorrectable memory error is detected.

Memory error data fields are described below.

E (Error type) — The type of memory error encountered, uncorrectable or correctable, is indicated in word 0, bits 2 and 3 of the Exchange Package. Bit 2 is set for an uncorrectable memory error; bit 3 is set for a correctable memory error.

S (Syndrome) — The 8 syndrome bits used in detecting a memory data error are returned in word 0, bits 4 through 11 of the Exchange Package. See section 2 for additional information.

R (Read mode) — This field indicates the read mode in progress when a memory data error occurred and is in word 1, bits 0 and 1 of the Exchange Package. These bits assume the following values:

00 I/O
01 Scalar (memory references with A or S)
10 Vector, B, or T
11 Instruction fetch or exchange

CSB (Read address) — The 10-bit CSB field contains the address where a memory data error occurred. Word 1, bits 7 through 11 (B) of the Exchange Package contain bits 2^4 through 2^0 of the address and can be considered as the bank address; word 1, bits 2 through 6 (CS) of the Exchange Package contain bits 2^{21} through 2^{17} (chip select) of the address.

VNU (Vector not used) The vector not used bit sets if during the execution intervals, no 076, 077 or 140 through 177 instructions issued. The VNU bit is in word 2, bit 0.

EXCHANGE REGISTERS

Three special registers are instrumental in the exchange mechanism: the Exchange Address (XA) register, the Mode (M) register, and the Flag (F) register. These three registers are described below.

Exchange Address register

The 8-bit Exchange Address (XA) register specifies the first word address of a 16-word Exchange Package loaded by an exchange operation. The register contains the high-order 8 bits of a 12-bit field specifying the address. The low-order bits of the field are always 0; an Exchange Package must begin on a 16-word boundary. The 12-bit limit requires that the absolute address be in the lower 4096 ($10{,}000_8$) 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 the beginning address (XA) in memory.

Mode register

The 10-bit Mode (M) register contains part of the Exchange Package for a currently active program. The M register bits are assigned in words 1 and 2 of the Exchange Package as follows.

Word 1

Bit	Description
35	Waiting for Semaphore (WS) flag; when set, the CPU exchanged when a test and set instruction was holding in the CIP register.
36	Floating-point Error Status (FPS) flag; when set, a floating-point error has occurred regardless of the state of the Floating-point Error Mode flag.
37	Bidirectional Memory Mode (BDM) flag; when set, block reads and writes can operate concurrently.

Word 1 (continued)

Bit	Description
38	Selected for external interrupts (SEI) flag; when set, this CPU is preferred for I/O interrupts.
39	Interrupt Monitor Mode (IMM) flag; when set, enables all interrupts in monitor mode except PC, MCU, I/O, and normal exit.

Word 2

Bit	Description
35	Operand Range Error Mode (IOR) flag; when set, enables interrupts on operand range errors.
36	Correctable Memory Error Mode (ICM) flag; when set, enables interrupts on correctable memory data errors.
37	Floating-point Error Mode (IFP) flag; when set, enables interrupts on floating-point errors.
38	Uncorrectable Memory Error Mode (IUM) flag; when set, enables interrupts on uncorrectable memory data errors.
39	Monitor Mode (MM) flag; when set, inhibits all interrupts except memory errors.

The 10 bits are set selectively during an exchange sequence.

Word 1, bit 37 (Bidirectional Memory Mode flag) can be set or cleared by using instructions 0026 (enable bidirectional memory transfers) and 0025 (disable bidirectional memory transfers).

Word 2, bit 35 (Operand Range Error Mode flag) can be set or cleared during the execution interval of a program by using instructions 0023 (enable interrupt on operand range error) and 0024 (disable interrupt on operand range error).

Word 2, bit 37 (Floating-point Error Mode flag), can be set or cleared during the execution interval for a program by using instructions 0021 (enable interrupt on floating-point error) and 0022 (disable interrupt on floating-point error).

Word 1, bits 36 and 37 and word 2, bits 35 and 37 can be read with instruction 073i01. Word 1, bits 35 and 36 indicate the state of the CPU at the time of the exchange. The remaining bits are not altered during the execution interval for the Exchange Package and can be altered only when the Exchange Package is inactive in storage.

Flag register

The 11-bit Flag (F) register contains part of the Exchange Package for the currently active program. This register is located in word 3 and contains 11 flags individually identified within the Exchange Package. Setting any of these flags interrupts program execution. When one or more flags are set, a Request Interrupt signal is sent to initiate an exchange sequence. The contents of the F register are stored along with the rest of the Exchange Package. The monitor program can analyze the eleven flags for the cause of the interruption. Before the monitor program exchanges back to the package, it must clear the flags in the F register area of the package. If any bit remains set, another exchange occurs immediately.

The F register bits are assigned in word 3 of the Exchange Package as follows.

Word 3

Bit	Description
14	Interrupt from Internal CPU (ICP) flag; set when the other CPU issues instruction 001401.
15	Deadlock (DL) flag; set when all CPUs in a cluster are holding issue on a test and set instruction.
31	Programmable Clock Interrupt (PCI) flag; set when the interrupt countdown counter in the programmable clock equals 0. The programmable clock is explained later in this section.
32	MCU Interrupt (MCU) flag; set when the MIOP sends this signal.
33	Floating-point Error (FPE) flag; set when a floating-point range error occurs in any of the floating-point functional units and the Enable Floating-point Interrupt flag is set. Floating-point functional units are explained in section 4, computation.
34	Operand Range Error (ORE) flag; set when a data reference is made outside the boundaries of the Data Base Address (DBA) and Data Limit Address (DLA) registers and the Enable Operand Range Interrupt flag is set. Operand range error is explained later in this section.

Word 3 (continued)

Bit	Description
35	Program Range Error (PRE) flag; set when an instruction fetch is made outside the boundaries of the Instruction Base Address (IBA) and Instruction Limit Address (ILA) registers. Program range error is explained later in this section.
36	Memory Error (ME) flag; set when a correctable or uncorrectable memory error occurs and the corresponding enable memory error mode bit is set in the M register.
37	I/O Interrupt (IOI) flag; set when a 6 Mbyte channel or the 1250 Mbyte channel completes a transfer.
38	Error Exit (EEX) flag; set by an error exit instruction (000).
39	Normal Exit (NEX) flag; set by a normal exit instruction (004).

Any flag (except the Memory Error flag) can be set in the F register only if the active Exchange Package is not in monitor mode. Such flags are set only if word 2, bit 39 of the M register is 0. Except for the Memory Error flag, if the program is in monitor mode and the conditions for setting an F register are present, the flag remains cleared and no exchange sequence is initiated.

Cluster Number register

The 2-bit Cluster Number (CLN) register determines the CPU's cluster. The contents of the CLN register are used to determine which set of SB, ST, and SM registers the CPU can access. If the CLN register is 0, then the CPU does not have access to any SB, ST, or SM register. The contents of the CLN registers in both CPUs are also used to determine the condition necessary for a deadlock interrupt.

Program State register

The content of the 1-bit Program State (PS) register is manipulated by the operating system to represent different program states in the CPUs concurrently processing a single program.

A registers

The current contents of all A registers are stored in bits 40 through 63 of word 0 through 7 during exchange.

S registers

The current contents of all S registers are stored in bits 4 through 11 of word 0 during exchange.

Program Address register

The contents of the Program Address (P) register (first program instruction not yet issued) are stored in bits 16 through 39 of word 0. This instruction is the first instruction to be issued when this program begins again.

ACTIVE EXCHANGE PACKAGE

An active Exchange Package resides in the operating registers. The interval of time when the Exchange Package and the program associated with it are active is called the execution interval. An execution interval begins with an exchange sequence where the subject Exchange Package moves from memory to the operating registers. An 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. At the same time, the exchange sequence moves 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. Location of this block is specified by the content of the XA register and is a part of the currently active Exchange Package. The exchange sequence can be initiated by deadstart sequence, interrupt flag set, or program exit.

Exchange initiated by deadstart sequence

The deadstart sequence forces the XA register content to 0 for both CPUs and also forces an interrupt in one CPU. These two actions cause an exchange using memory address 0 as the location of the Exchange Package. The inactive Exchange Package at address 0 then moves into the operating

registers and initiates a program using these parameters. The Exchange Package swapped to address 0 is largely indeterminate because of the deadstart operation. New data entered at these storage addresses then discards the old Exchange Package in preparation for starting the second CPU with an interprocessor interrupt. When instruction 001401 (IP) is issued in the first CPU, the second CPU exchanges to address 0 in memory. (The second CPU can be deadstarted first by selecting a switch on the control panel.)

Exchange initiated by interrupt flag set

An exchange sequence can be initiated by setting any one of the interrupt flags in the F register. Setting of one or more flags causes a Request Interrupt signal to initiate an exchange sequence.

Exchange initiated by program exit

Two program exit instructions initiate an exchange sequence. Timing of the instruction execution is the same in either case, the difference is determined by which of the two flags is set in the F register. The two instructions are:

000	ERR	Error exit
004	EX	Normal exit

The two exits enable a program to request its own termination. A non-monitor (object) program usually uses the normal exit instruction to exchange back to the monitor program. The error exit allows for abnormal termination of an object program. The exchange address selected is the same as for a normal exit.

Each instruction has a flag in the F register. The appropriate flag is set if 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. Flags are checked for evaluation of the program termination cause.

The monitor program selects an inactive Exchange Package for activation by setting the address of the inactive Exchange Package in the XA register and then executing a normal exit instruction.

Exchange sequence issue conditions

The following are hold issue conditions, execution time, and special cases for an exchange sequence.

Hold conditions:

- NIP register contains a valid instruction
- S, V, or A registers busy

Execution time:

For 32 banks, 40 CPs; consists of an exchange sequence (24 CPs) and a fetch operation (16 CPs).

For 16 banks, 42 CPs; consists of an exchange sequence (24 CPs) and a fetch operation (18 CPs).

Special cases:

If a test and set instruction is holding in the CIP register, both CIP and NIP registers are cleared and the exchange occurs with the WS (Waiting for Semaphore) flag set and the P register pointing to the test and set instruction.

EXCHANGE PACKAGE MANAGEMENT

Each 16-word Exchange Package resides in an area defined during system deadstart. The defined area must lie within the lower 4096 ($10,000_8$) words of memory. The package at address 0 is the deadstart monitor program's Exchange Package. Other packages provide for object programs and monitor tasks. Non-monitor 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 Exchange Package areas. The defined field allows the monitor program to define or alter all Exchange Packages other than its own when it is the currently active Exchange Package. Since no interlock exists between an exchange sequence in a CPU and memory transfers in another CPU, modification of Exchange Packages which can be used by another CPU should be avoided, except under software controlled situations.

Proper management of Exchange Packages dictates that a non-monitor program always exchanges back to the monitor program that exchanged to it. The exchange ensures that the program information is always exchanged into its proper Exchange Package.

For example, the monitor program (A) begins an execution interval following deadstart. No interrupts (except memory) can terminate its execution interval since it is in monitor mode. Program A voluntarily exits by issuing a normal exit instruction (004). However, before doing so, program A sets the contents of the XA register to point to the user

program (B) Exchange Package so that program B is the next program to execute. Program A sets the exchange address in program B's Exchange Package to point back to program A.

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

To illustrate the exchange sequence, assume that while program B is executing, an interrupt flag sets initiating an exchange sequence. Since program B cannot alter the XA register, the exit is back to program A. Program B's parameters exchange back into its Exchange Package area; program A's parameters held in program B's package during the execution interval exchange back into the operating registers.

Program A, upon resuming execution, determines an interrupt has caused the exchange and sets the XA register to call the proper interrupt processor into execution. To do this, program A sets XA to point to the Exchange Package for the interrupt processing program (C). Program A clears the interrupt and initiates execution of program C by executing a normal exit instruction (004). Depending on the operating task, program C can execute in monitor mode or in user mode.

Further information on Exchange Package management is contained in the COS EXEC/STP/CSP Internal Reference Manual, publication SM-0040.

MEMORY FIELD PROTECTION

At execution time each object program has a designated field of memory for instructions and data. The field limits are specified by the monitor program when the object program is loaded and initiated. The fields can begin at any word address that is a multiple of 32 (that is, 40_8) and can continue to another address that is one less than a multiple of 32. The fields can overlap.

All memory addresses contained in the object program code are relative to one of the two base addresses specifying the beginning of the appropriate field. An object program cannot read or alter any memory location with an absolute address lower than that base address. Each object program reference to memory is checked against the limit and base addresses to determine if the address is within the bounds assigned. A memory read

reference beyond the assigned field limits issues and completes, but a zero value is transferred from memory. A memory write reference beyond the assigned field limits is allowed to issue, but no write occurs.

Field limits are contained in four registers: the Instruction Base Address (IBA) register, the Instruction Limit Address (ILA) register, the Data Base Address (DBA) register, and the Data Limit Address (DLA) register. These four registers are described below.

INSTRUCTION BASE ADDRESS REGISTER

The 17-bit Instruction Base Address (IBA) register holds the base address of the user's instruction field. An instruction can only be executed by the CPU if the absolute address at which the instruction is located is greater than or equal to the contents of the current Exchange Package IBA register of the program executing. This determination is made at instruction buffer fetch time by the CPU. The contents of this register are interpreted as the high-order 17 bits of a 22-bit memory address. The low-order 5 bits of the address are assumed to be zero because of the instruction buffer length, 32 decimal words. Absolute memory addresses for an instruction fetch are formed by adding the IBA register to the P register (high-order 22 bits) modulo two to the twenty-second power. A reference to an absolute address less than the address defined by IBA can only occur via a jump or branch instruction to an address beyond the memory capacity of the machine.

INSTRUCTION LIMIT ADDRESS REGISTER

The 17-bit Instruction Limit Address (ILA) register holds the limit address of the user's field. An instruction can only be executed by the CPU if the absolute address where it is located is less than the contents of the current Exchange Package ILA register of the program executing. This determination is made at instruction buffer fetch time by the CPU. The contents of the register are interpreted as the high-order 17 bits of a 22-bit memory address. The low-order 5 bits of the address are assumed to be zero because of the instruction buffer length, 32 decimal words. Absolute memory addresses for an instruction fetch are formed by adding the IBA register to the P register (high-order 22 bits) modulo two to the twenty-second power. The largest absolute address that can be executed by a program is defined by $[(ILA) \times 2^5] - 1$.

If the final absolute address of the instruction buffer fetch as computed by the CPU does not fall between the range of addresses contained within the currently executing Exchange Package IBA and ILA registers, the CPU generates a Program Range Error Interrupt.

DATA BASE ADDRESS REGISTER

The 17-bit Data Base Address (DBA) register holds the base address of the user's data field. An operand can only be fetched or stored by the CPU if the absolute address where the operand is located is greater than or equal to the contents of the current Exchange Package DBA register of the program executing. This determination is made each time an operand is fetched or stored by the CPU. The contents of the register are interpreted as the high-order 17 bits of a 22-bit memory address. The low-order 5 bits of the register are assumed to be zero. Absolute memory addresses for operands are formed by adding the DBA register to the modified operand address modulo two to the twenty-second power.

DATA LIMIT ADDRESS REGISTER

The 17-bit Data Limit Address (DLA) register holds the (upper) limit address of the user's data field. An operand can only be fetched or stored by the CPU if the absolute address where the operand is located is less than the contents of the current Exchange Package DLA register of the program executing. This determination is made each time an operand is fetched or stored by the CPU. The contents of the register are interpreted as the high-order 17 bits of a 22-bit memory address. The low-order 5 bits of the register are assumed to be zero. Absolute memory addresses for operands are formed by adding the DBA register to the modified operand address modulo two to the twenty-second power. The largest absolute address that can be referenced for data by a program is defined by $[(DLA) \times 2^5] - 1$.

If the final absolute address of the operand as computed by the CPU does not fall between the range of addresses contained within the currently executing Exchange Package DBA and DLA registers, the CPU generates an Operand (address) Range Error Interrupt.

PROGRAM RANGE ERROR

The Program Range Error flag sets if a memory reference outside the boundaries of the IBA and ILA register is for an instruction fetch. An out-of-range memory reference can occur in a non-monitor mode program on a branch or jump instruction calling for a program address above or below the limits. The Program Range Error flag causes an error condition that terminates program execution. The monitor program checks the state of the Program Range Error flag and takes appropriate action, perhaps aborting the user program.

OPERAND RANGE ERROR

The Operand Range Error flag sets if the Operand Range Error Mode is set and a memory reference outside the boundaries of the DBA and DLA registers is called to read or write an operand for an A, B, S, T, or V register and the operand range interrupt flag is set. The Operand Range Error flag causes an error condition that terminates the user program execution. The monitor program checks the state of the Operand Range Error flag and takes appropriate action, perhaps aborting the user program.

PROGRAMMABLE CLOCK

The programmable clock can be used to accurately measure the duration of intervals. Intervals selected under monitor program control generate a periodic interrupt. The clock frequency is 105 Mhz. Intervals from 9.5 nanoseconds to approximately 40.8 seconds are possible. Intervals shorter than 100 microseconds are not practical due to the monitor overhead involved in processing the interrupt. Supporting the programmable clock are the Interrupt Interval (II) register, the Interrupt Countdown (ICD) counter, and four monitor mode instructions.

INSTRUCTIONS

Four monitor mode instructions support the programmable clock:

0014j4	PCI Sj	Enter Interrupt Interval (II) register with (Sj)
001405	CCI	Clear the programmable clock interrupt request
001406	ECI	Enable the programmable clock interrupt request
001407	DCI	Disable the programmable clock interrupt request

INTERRUPT INTERVAL REGISTER

The 32-bit Interrupt Interval (II) register can be loaded with a binary value equal to the number of CPs that are to elapse between programmable clock interrupt requests. The interrupt interval is transferred from the low-order 32 bits of the Sj register into the II register and the ICD counter when instruction 0014j4 is executed.

This value is held in the II register and is transferred to the ICD counter each time the counter reaches 0 and generates an interrupt request. The content of the II register is changed only by another instruction 0014*j*4.

INTERRUPT COUNTDOWN COUNTER

The 32-bit Interrupt Countdown (ICD) counter is preset to the contents of the II register when instruction 0014*j*4 is executed. This counter runs continuously but counts down, decrementing by 1 each CP until the content of the counter is 0. The ICD sets the programmable clock interrupt request and samples the interval value held in the II register. The ICD repeats the countdown to 0 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 clear programmable clock interrupt request is executed. A programmable clock interrupt request can be set only after the enable programmable clock interrupt request is executed. A programmable clock interrupt request causes an interrupt only 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 can be cleared by executing instruction 001405.

Following any deadstart, the monitor program should ensure the state of the programmable clock interrupt by issuing instructions 001405 and 001407.

DEADSTART SEQUENCE

The deadstart sequence of operations starts a program running in the CRAY X-MP mainframe after power has been turned off and then turned on again or whenever the operating system is to be re-initialized in the mainframe. All registers in the machine, all control latches, and all words in memory should be considered invalid after power has been turned on. The following sequence of operations to begin the program is initiated by the I/O Subsystem.

1. Turn on Master Clear signal.

2. Turn on I/O Clear signal.

3. Turn off I/O Clear signal.

4. Load memory via I/O Subsystem.

5. Turn off Master Clear signal.

The Master Clear signal halts all internal computation and forces critical control latches to predetermined states. The I/O Clear signal clears the input channel address register of the MCU channel and activates the MCU input channel. All other input channels remain inactive. The I/O Subsystem then loads an initial Exchange Package and monitor program. The Exchange Package must be located at address 0 in memory. Turning off the Master Clear signal initiates the exchange sequence to read this package and to begin execution of the monitor program in CPU 0 (PN = 0). CPU 1 (PN = 1) remains in a master cleared state until instruction 001401 (IP) is issued in CPU 0. Then CPU 1 exchanges to address 0 in memory.

Because the exchange of CPU 0 overwrites the contents of the inactive Exchange Package at address 0, CPU 0 must reinitialize the Exchange Package at address 0 before allowing CPU 1 to start. (CPU 1 can be started first by selecting a switch on the control panel.) Subsequent actions are dictated by the design of the operating system.

CPU COMPUTATION SECTION 4

INTRODUCTION

Each CPU contains an identical independent computation section. A computation section consists of operating registers and functional units associated with three types of processing: address, scalar, and vector. Address processing operates on internal control information such as addresses and indexes and has two levels of 24-bit registers and two integer arithmetic functional units. Scalar and vector processing are performed on data.

A vector is an ordered set of elements. A vector instruction operates on a series of elements repeating the same function and producing a series of results. Scalar processing starts an instruction, handles one operand or operand pair, then stops the operation.

The main advantage of vector over scalar processing is eliminating instruction start-up time for all but the first operand. Scalar processing has two levels of 64-bit scalar registers, four functional units dedicated solely to scalar processing, and three floating-point functional units shared with vector operations. Vector processing has a set of 64-element registers of 64 bits each, four functional units dedicated solely to vector applications, and three floating-point functional units supporting both scalar and vector operations.

Address information flows from Central Memory or from control registers to address registers. Information in the address registers is 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 Central Memory from the address registers.

Data flow in a computation section is generally from Central Memory to registers and from registers to functional units. Results flow from functional units to registers and from registers to Central Memory or back to functional units. Data flows along either the scalar or vector path depending on the processing mode. An exception is that scalar registers can provide one required operand for vector operations performed in the vector functional units.

Integer or floating-point arithmetic operations are performed in the computation section. Integer arithmetic is performed in twos complement mode. Floating-point quantities have signed magnitude representation.

Floating-point instructions provide for addition, subtraction, multiplication, and reciprocal approximation. The reciprocal approximation instructions provide for a floating-point divide operation using a multiple instruction sequence. These instructions produce 64-bit results (1-bit sign, 15-bit exponent, and 48-bit normalized coefficient).

Integer or fixed-point operations are integer addition, integer subtraction, and integer multiplication. Integer addition and subtraction operations produce either 24-bit or 64-bit results. An integer multiply operation produces a 24-bit result. A 64-bit integer multiply operation is done through a software algorithm using the floating-point multiply functional unit to generate multiple partial products. These partial products are then shifted and merged to form the full 64-bit product. No integer divide instruction is provided; the operation is accomplished through a software algorithm using floating-point hardware.

The instruction set includes Boolean operations for OR, AND, equivalence, 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, most operations are implemented in vector and 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 can be positive or negative in either mode. Indexing allows matrix operations in vector mode to be performed on rows or the diagonal as well as conventional column-oriented operations.

Population and parity counts are provided for both vector and scalar operations. An additional scalar operation is the leading zero counts.

Characteristics of a CPU computation section are summarized below.

- Integer and floating-point arithmetic
- Twos complement integer arithmetic
- Signed magnitude floating-point arithmetic
- Address, scalar, and vector processing modes
- Thirteen 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

OPERATING REGISTERS

Operating registers, a primary programmable resource of a CPU, enhances the speed of the system by satisfying heavy demands for data made by the functional units. A single functional unit can require one to three operands per clock period (CP) to perform the necessary functions and can deliver results at a rate of one per CP. Multiple functional units can be used concurrently.

A CPU has three primary and two intermediate sets of registers. The primary sets of registers are address, scalar, and vector designated in this manual as A, S, and V, respectively. These registers are considered primary because functional units can access them directly.

For the address and scalar registers, an intermediate level of registers exists which is not accessible to the functional units but acts as a buffer for the primary registers. Block transfers are possible between these registers and Central Memory so that the number of memory reference instructions required for scalar and address operands is greatly reduced. The intermediate registers that support the address registers are referred to as B registers. The intermediate registers that support scalar registers are referred to as T registers.

ADDRESS REGISTERS

Figure 4-1 illustrates registers and functional units used for address processing. The two types of address registers are designated A registers and B registers and are described in the following paragraphs.

A REGISTERS

Eight 24-bit A registers serve a variety of applications but are primarily used as address registers for memory references and as index registers. They provide values for shift counts, loop control, and channel I/O operations and receive values of population count and leading zeros count. In address applications, A registers index the base address for scalar memory references and to provide both a base address and an address increment 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 by delivering the results to A registers.

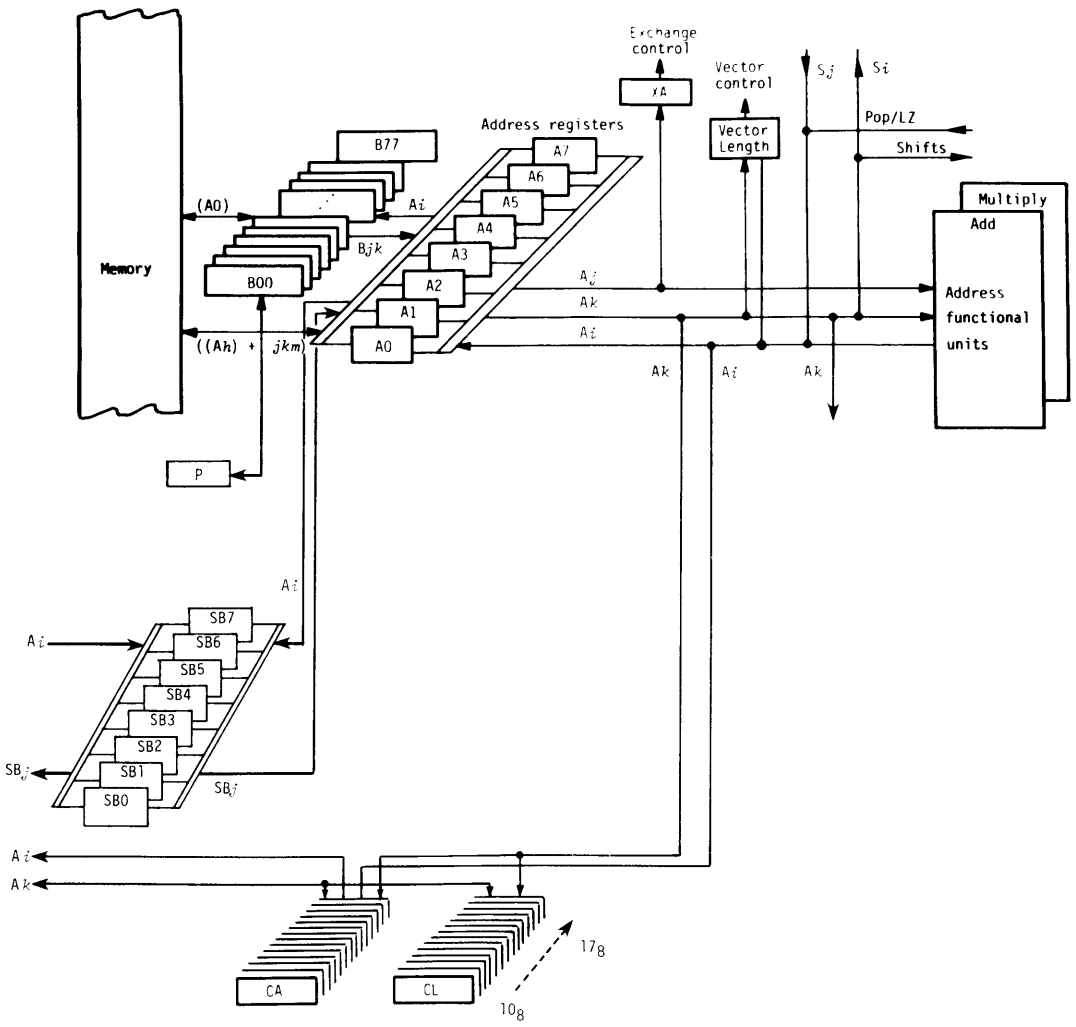


Figure 4-1. Address registers and functional units

Data is moved directly between Central Memory and A registers or is placed in B registers. Placing data in B registers allows buffering of the data between A registers and Central Memory. Data can also be transferred between A and S registers and between A and Shared Address (SB) registers.

The Vector Length (VL) register and Exchange Address (XA) register are set by transmitting a value to them from an A register. The VL register can also be transmitted to an A register. (The VL register is described under Vector Control Registers later in this section.)

When an issued instruction delivers new data to an A register, a reservation is set for that register. The reservation prevents issue of instructions that use the register until the new data is delivered.

In this manual, the A registers are individually referred to by the letter A followed by a number ranging from 0 through 7. Instructions reference A registers by specifying the register number as the *h*, *i*, *j*, or *k* designator as described in section 5.

The only register implicitly referenced is the A0 register as illustrated in the following instructions:

010*ijkm*	JAZ *exp*	Branch to *ijkm* if (A0)=0
011*ijkm*	JAN *exp*	Branch to *ijkm* if (A0)≠0
012*ijkm*	JAP *exp*	Branch to *ijkm* if (A0) is positive, includes (A0)=0
013*ijkm*	JAM *exp*	Branch to *ijkm* if (A0) is negative
034*ijk*	B*jk*,A*i* ,A0	Read (A*i*) words to B register *jk* from (A0)
035*ijk*	,A0 B*jk*,A*i*	Store (A*i*) words at B register *jk* to (A0)
036*ijk*	T*jk*,A*i* ,A0	Read (A*i*) words to T register *jk* from (A0)
037*ijk*	,A0 T*jk*,A*i*	Store (A*i*) words at T register *jk* to (A0)
176*i*0*k*	V*i* ,A0,A*k*	Read (VL) words to V*i* from (A0) incremented by (A*k*)
1770*jk*	,A0,A*k* V*j*	Store (VL) words from V*i* from (A0) incremented by (A*k*)

Section 5 of this manual contains additional information on the use of A registers by instructions.

B REGISTERS

A computation section contains sixty-four 24-bit B registers used as intermediate storage for the A registers. Typically, B registers contain data to be referenced repeatedly over a sufficiently long span making it unnecessary to retain the data in either A registers or in Central Memory. Examples of uses are loop counts, variable array base addresses, and dimensions.

Transfer of a value between an A register and a B register requires only 1 CP. A block of B registers can be transferred to or from Central Memory at the maximum rate of one 24-bit value per CP. A reservation is made on all B registers during block transfers to and from B registers.

NOTE

Other instructions can issue on the CRAY X-MP while a block of B registers is being transferred to or from Central Memory.

In this manual, B registers are individually referred to by the letter B followed by a 2-digit octal number ranging from 00 through 77. Instructions reference B registers by specifying the B register number in the *jk* designator as described in section 5.

The only B register implicitly referenced is the B00 register. On execution of the return jump instruction (007), register B00 is set to the next instruction parcel address (P) and a branch to an address specified by *ijkm* occurs. Upon receiving control, the called routine conventionally saves (B00) so that the B00 register is available 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 instruction 0050*jk*. Conventionally, this instruction, which is a branch to (B*jk*), causes the address saved in B*jk* to be entered into the P register as the address of the next instruction parcel to be executed.

SCALAR REGISTERS

Figure 4-2 illustrates registers and functional units used for scalar processing. The two types of scalar registers are designated S registers and T registers and are described in the following paragraphs.

S REGISTERS

Eight 64-bit S registers are the principal scalar registers for a CPU serving as the source and destination for operands executing scalar arithmetic and logical instructions. Scalar functional units perform both integer and floating-point arithmetic operations.

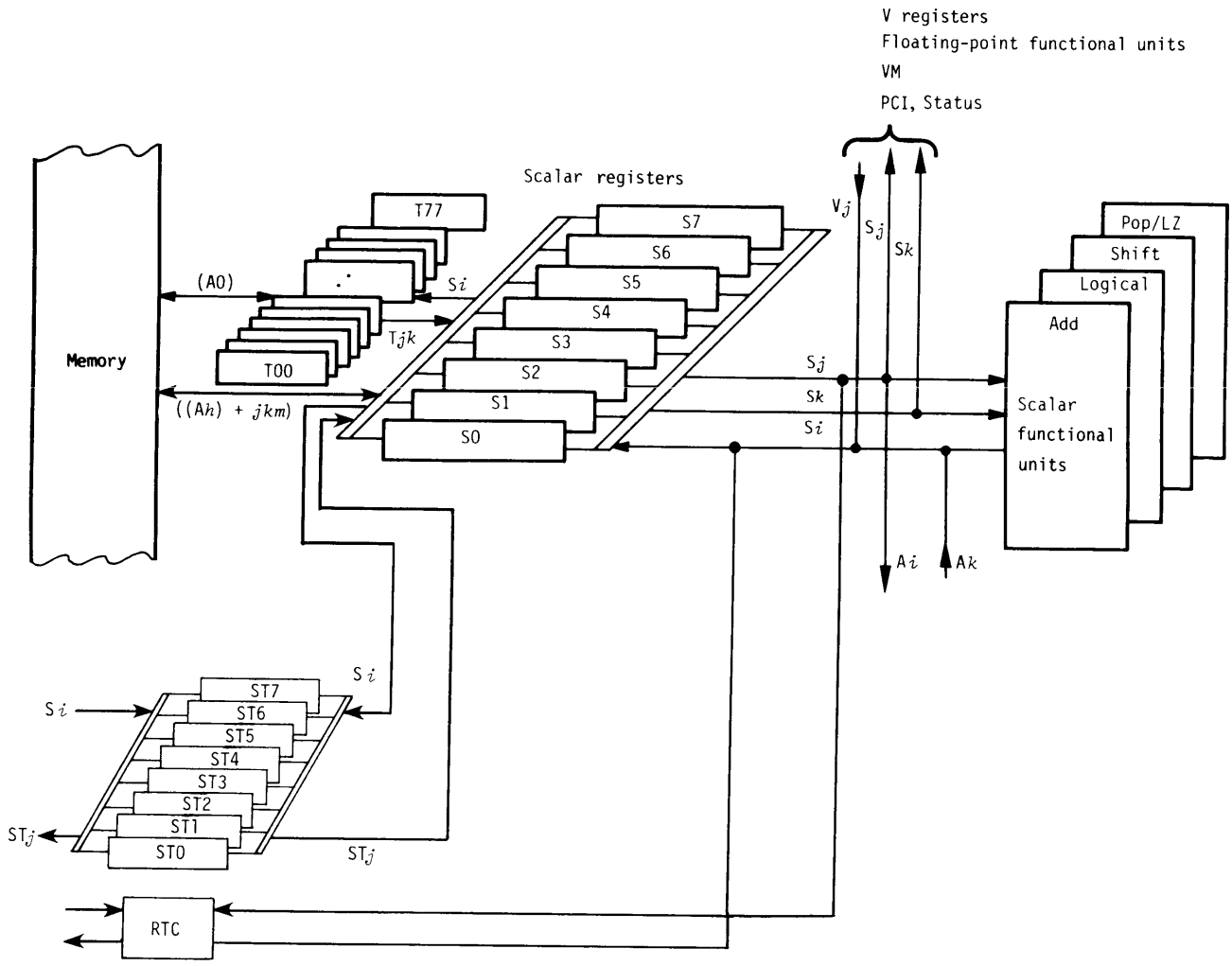


Figure 4-2. Scalar registers and functional units

S registers can 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 is moved directly between Central Memory and S registers or is placed in T registers. This intermediate step allows buffering of scalar operands between S registers and Central Memory. Data is also transferred between A and S registers, between S and Shared Scalar (ST) registers, and between S and Semaphore (SM) registers.

Other uses of the S registers are the setting or reading of the Vector Mask (VM) register or the Real-time Clock (RTC) register or setting the Interrupt Interval (II) register.

When an issuing instruction delivers new data to an S register, a reservation is set for that register preventing issue of instructions that read the register until the new data is delivered.

In this manual, the S registers are individually referred to by the letter S followed by a number ranging from 0 through 7. Instructions reference S registers by specifying the register number as the *i*, *j*, or *k* designator as described in section 5.

The only register implicitly referenced is the S0 register as illustrated in the following instructions.

014*ijkm*	JSZ	*exp*	Branch to *ijkm* if (S0)=0
015*ijkm*	JSN	*exp*	Branch to *ijkm* if (S0)≠0
016*ijkm*	JSP	*exp*	Branch to *ijkm* if (S0) is positive, includes (S0)=0
017*ijkm*	JSM	*exp*	Branch to *ijkm* if (S0) is negative
052*ijk*	S0	S*i* *exp*	Shift (S*i*) left *jk* places to S0
053*ijk*	S0	S*i* *exp*	Shift (S*i*) right *jk* places to S0

The 8-bit Status register provides the status of the following flags:

- Processor Number (PN)
- Program State (PS)
- Cluster Number (CN)
- Floating-point Interrupts Enabled (IFP)
- Floating-point Error (FPS)
- Bidirectional Memory Enabled (BDM)
- Operand Range Interrupts Enabled (IOR)

Instruction 073 sends the contents of the Status register to an S register.

Section 5 of this manual has additional information on the use of S registers by instructions.

T REGISTERS

The computation section has sixty-four 64-bit T registers used as intermediate storage for the S registers. Data is transferred between T and S registers and between T registers and Central Memory. Transfer of a value between a T register and an S register requires only 1 CP. T registers reference Central Memory through block read and block write instructions. Block transfers occur at a maximum rate of one word per CP. A reservation is made on all T registers during block transfers to and from T registers.

NOTE

Other instructions can issue on the CRAY X-MP while a block of T registers is being transferred to or from Central Memory.

In this manual, T registers are referred to by the letter T and a 2-digit octal number ranging from 00 through 77. Instructions reference T registers by specifying the octal number as the *jk* designator as described in section 5.

VECTOR REGISTERS

Figure 4-3 illustrates the registers and functional units used for vector operations. Vector registers and vector control registers are described in the following paragraphs.

V REGISTERS

The major computational registers of a CPU are eight V registers, each with 64 elements. Each V register element has 64 bits. When associated data is grouped into successive elements of a V register, the register quantity can be treated as a vector. Examples of vector quantities are rows or columns of a matrix or elements of a table. Computational efficiency is achieved by identically processing each element of a vector. Vector instructions provide for the iterative processing of successive V register elements. A vector operation always begins when operands are obtained from the first element of the operand V registers and the result is delivered to the first element of a V register. Successive elements are provided each CP and as each operation is performed, the result is delivered to successive elements of the result V

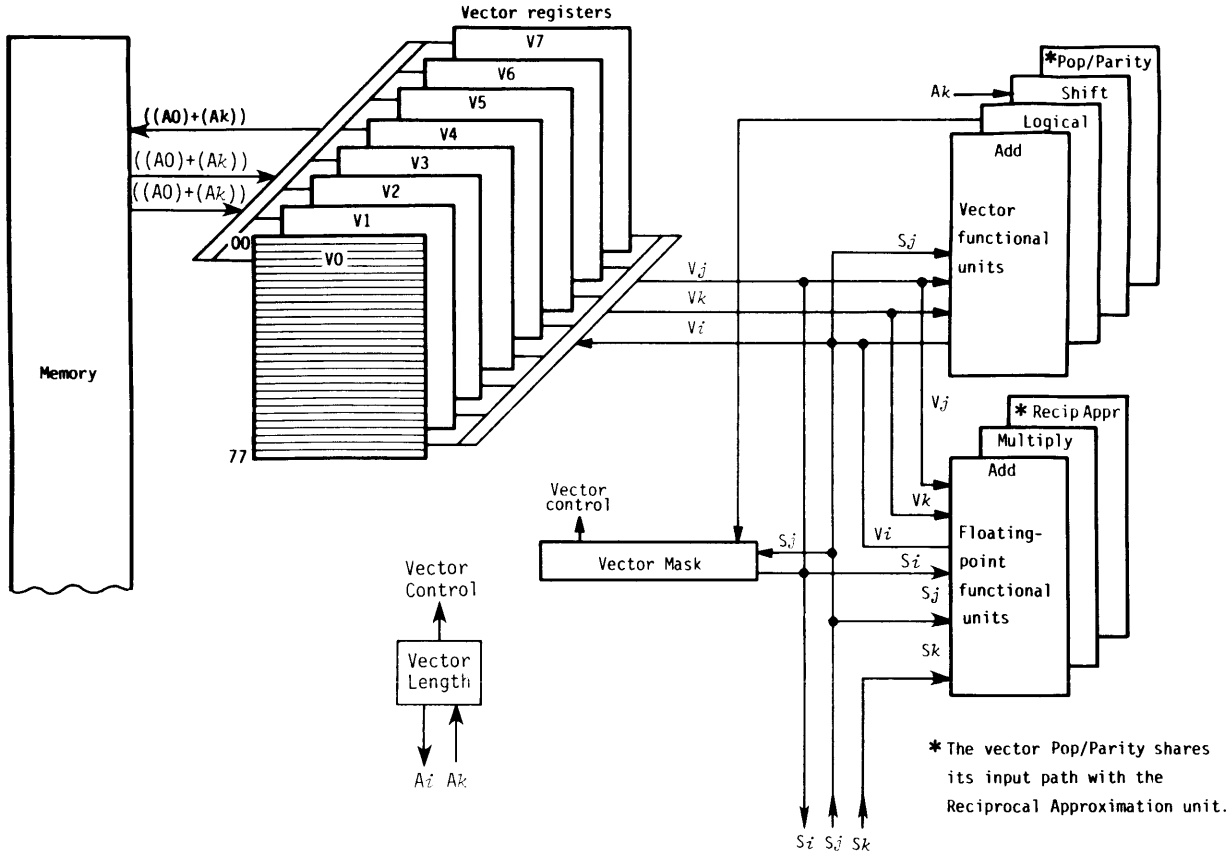


Figure 4-3. Vector registers and functional units

register. The vector operation continues until the number of operations performed by the instruction equals a count specified by the content of the VL register.

Contents of a V register are transferred to or from Central Memory in a block mode by specifying a first word address in Central Memory, an increment or decrement for the Central Memory address, 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 CP, depending upon bank conflicts. Discontinuities in the vector data stream can occur as a result of memory conflicts. These discontinuities, although not inhibiting chained operations, can appear in the chained operation data stream. Any discontinuity in the data stream adds proportionally to the total execution time of the vector operation.

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

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 can choose to construct the vector loop code in a number of ways. The processing of long vectors in FORTRAN is handled by the compiler and is transparent to the programmer.

A V register receiving results can also supply operands to a subsequent operation. Using a register as both a result and operand register in two different operations allows for the chaining together of two or more vector operations and two or more results can be produced per CP. Chained operations are detected automatically by a CPU and are not explicitly specified by the programmer. A programmer can reorder certain code segments to gain as much concurrency as possible in chained operations.

A conflict can occur between vector and scalar operations involving floating-point operations and memory access. With the exception of these operations, the functional units are always available for scalar operations. A vector operation occupies the selected functional unit until the vector is processed.

Parallel vector operations can be processed in two ways:

- Using different functional units and all different V registers
- Using the result stream from one V register simultaneously as the operand to another operation using a different functional unit (chain mode)

Parallel operations on vectors allow the generation of two or more results per CP. Most vector operations use two V registers as operands or one S and one V register as operands. Exceptions are vector shifts, vector reciprocal, and the load or store instructions.

In this manual, the V registers are individually referred to by the letter V followed by a number ranging from 0 through 7. Vector instructions reference V registers by specifying the register number as the i, j, or k designator as described in section 5.

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

NOTE

Parallel loading and storing of V registers is possible; two load operations and one store operation can occur simultaneously.

V register reservations and chaining

Reservation describes the condition of a register in use; that is, the register is not available for another operation as a result or as an operand register. Each register has two reservation conditions, one reserving it as a operand register and one reserving it as a result register. 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.

If a V register is reserved as a result and not as an operand, it can be used at any time as an operand and chaining occurs. This flexible chaining mechanism allows chaining to begin at any point in the result vector data stream. Full chaining occurs if the instruction causing chaining is issued before or at the time element 0 of the result arrives at the V register. Partial chaining occurs if the instruction issues after the arrival of element 0. Thus, the amount of concurrency in a chained operation depends upon the relationship between the issue time of the chaining instruction and the result vector data stream.

If a V register is reserved as an operand, it cannot be used as a result or operand register until the operand reservation clears. However, a V register can be used as both an operand and result in the same vector operation. A V register can serve only one vector operation as the source of one or both operands. A V register can serve only one vector operation as a result.

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. The S register can 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 and S register. Vector operations employing different lengths can proceed concurrently.

The A0 and Ak registers in a vector memory reference are treated similarly and are available for modification immediately after use.

****

CAUTION

Cray Research cautions against using a vector register as both a result and an operand if compatibility between a CRAY-1 and a CRAY X-MP is necessary because vector recursion is not available on all Cray Research, Inc., computers.

****

VECTOR CONTROL REGISTERS

The Vector Length (VL) register and Vector Mask (VM) register provide control information needed in the performance of vector operations and are described below.

Vector Length register

The 7-bit Vector Length (VL) register is set to 1 through 100_8 (VL = 0 gives VL = 100_8) specifying the length of all vector operations performed by vector instructions and the length of the vectors held by the V registers. The VL register controls the number of operations performed for instructions 140 through 177 and is set to an A register value using instruction 0020 or read using instruction 023i01.

Vector Mask register

The Vector Mask (VM) register has 64 bits, each corresponding to a word element in a V register. Bit 2^{63} corresponds to element 0, bit 2^0 to element 63. The mask is used with vector merge and test instructions to allow operations to be performed on individual vector elements.

The VM register can be set from an S register through instrucion 003 or can be created by testing a V register for a condition using instruction 175. The mask controls element selection in the vector merge instructions (146 and 147). Instruction 073 sends the contents of the VM register to an S register.

FUNCTIONAL UNITS

Instructions other than simple transmits or control operations are performed by specialized hardware known as functional units. Each unit implements an algorithm or a portion of the instruction set. Functional units have independent logic except for the Reciprocal Approximation and Vector Population Count units (described later in this section), which share some logic. All 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. Functional units operate essentially in 3-address mode with source and destination addressing limited to register designators.

All functional units perform algorithms in a fixed amount of time; delays are impossible once the operands have been delivered to the unit. Time required from delivery of the operands to the functional unit until completion of the calculation is called the functional unit time and is measured in 9.5-nanosecond CPs.

Functional units are fully segmented. This means a new set of operands for unrelated computation can enter a functional unit each CP even though the functional unit time can be more than 1 CP. This segmentation is possible when information arrives at the functional unit and is held in the functional unit or moves within the functional unit at the end of every CP.

Thirteen functional units are identified in this manual and are arbitrarily described in four groups: address, scalar, vector, and floating-point. Each of the first three groups functions with one of the primary register types (A, S, and V) to support the address, scalar, and vector modes of processing available in the CRAY X-MP. The fourth group, floating-point, supports either scalar or vector operations and accepts operands from or delivers results to S or V registers. In addition, Central Memory acts like a fourteenth functional unit for vector operations.

ADDRESS FUNCTIONAL UNITS

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 twos complement.

Address Add functional unit

The Address Add functional unit performs 24-bit integer addition and subtraction. The unit executes instructions 030 and 031. Addition and subtraction are performed in a similar manner. The twos complement subtraction for instruction 031 occurs when the ones complement of the A*k* operand is added to the A*j* operand. Then a 1 is added in the low-order bit position of the result. No overflow is detected in the Address Add functional unit.

The Address Add functional unit time is 2 CPs.

Address Multiply functional unit

The Address Multiply functional unit executes instruction 032 forming a 24-bit integer product from two 24-bit operands. No rounding is performed. The result consists of the least significant 24 bits of the product.

This functional unit is designed to handle address manipulations not exceeding its data capabilities. The programmer must be careful when multiplying integers in the functional unit because the unit does not detect overflow of the product and significant portions of the product could be lost.

The Address Multiply functional unit time is 4 CPs.

SCALAR FUNCTIONAL UNITS

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 functional unit which delivers its 7-bit result to an A register.

Four functional units are exclusively associated with scalar operations and are described below. Three functional units are used for both scalar and vector operations and are described in the section on floating-point functional units.

Scalar Add functional unit

The Scalar Add functional unit performs 64-bit integer addition and subtraction and executes instructions 060 and 061. The addition and subtraction are performed in a similar manner. The twos complement

subtraction for instruction 061 occurs when the ones complement of the Sk operand is added to the Sj operand. Then a 1 is added in the low-order bit position of the result. No overflow is detected in the Scalar Add functional unit.

The Scalar Add functional unit time is 3 CPs.

Scalar Shift functional unit

The Scalar Shift functional 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 nonzero.

The Scalar Shift functional unit executes instructions 052 through 057. Single-shift instructions (052 through 055) have a functional unit time of 2 CPs. Double-shift instructions (056 and 057) have a functional unit time of 3 CPs.

Scalar Logical functional unit

The Scalar Logical functional unit performs bit-by-bit manipulation of 64-bit quantities obtained from S registers. It executes instructions 042 through 051, the mask, and Boolean instructions. Instructions 042 through 051 have a functional unit time of 1 CP.

Scalar Population/Parity/Leading Zero functional unit

This functional unit executes instructions 026 and 027. Instruction 026ij0 counts the number of bits in an S register having a value of 1 in the operand and has a functional unit time of 4 CPs. Instruction 026ij1 returns a 1-bit population parity count (even parity) of the Sj register's contents. Instruction 027 counts the number of bits of 0 preceding a 1 bit in the operand and has a functional unit time of 3 CPs. For these instructions, the 64-bit operand is obtained from an S register and the 7-bit result is delivered 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, Shift, and Population/Parity functional units, which require only one operand, are exceptions. Results from a vector functional unit are delivered to a V register.

Successive operand pairs are transmitted each CP to a functional unit. The corresponding result emerges from the functional unit n CPs later, where n is the functional unit time and is constant for a given functional unit. The VL register determines the number of operand pairs to be processed by a functional unit.

Four functional units described in this section are exclusively associated with vector operations. 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 functional unit is used for a vector operation, the general description of vector functional units given in the subsection applies.

Vector functional unit reservation

A functional unit engaged in a vector operation remains busy during each CP and cannot participate in other operations. In this state, the functional unit is reserved. Other instructions requiring 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. A vector functional unit is reserved for (VL) + 4 CPs.

Vector Add functional unit

The Vector Add functional unit performs 64-bit integer addition and subtraction for a vector operation and delivers the results to elements of a V register. The unit executes instructions 154 through 157. Addition and subtraction are performed in a similar manner. For subtraction operations (156 and 157), the Vk operand is complemented before addition and a 1 is added into the low-order bit position of the result. No overflow is detected by the unit.

The Vector Add functional unit time is 3 CPs.

Vector Shift functional unit

The Vector Shift functional 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 and are end off with zero fill.

All shift counts are considered positive unsigned integers. If any bit higher than 2^6 is set, the shifted result is all zeros.

The Vector Shift functional unit executes instructions 150 through 153. The functional unit time is 4 CPs for instruction 152, and the functional unit time is 3 CPs for instructions 150, 151, and 153.

Vector Logical functional unit

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

The Vector Logical functional unit time is 2 CPs.

Vector Population/Parity functional unit

The Vector Population/Parity functional unit counts the 1 bits in each element of the source V register. The total number of 1 bits is the population count. This population count can be an odd or an even number, as shown by its low-order bit.

Instructions 174*ij*1 (vector population count) and 174*ij*2 (vector population count parity) use the same operation code as the vector reciprocal approximation instruction. Some restrictions for the Reciprocal Approximation functional unit also apply for vector population instructions (see subsection on Reciprocal Approximation). The vector population count instruction delivers the total population count to elements of the destination V register.

The vector population count parity instruction delivers the low-order bit of the count to the destination V register. The Vector Population/Parity functional unit time is 6 CPs.

FLOATING-POINT FUNCTIONAL UNITS

Three floating-point functional units perform floating-point arithmetic for scalar and vector operations. When executing a scalar instruction, operands are obtained from S registers and results are delivered to an S register. When executing most vector instructions, operands are obtained from pairs of V registers or from an S register and a V register. Results are delivered to a V register. An exception is the reciprocal approximation unit requiring only one input operand.

Information on floating-point out-of-range conditions is contained in the subsection on floating-point arithmetic.

Floating-point Add functional unit

The Floating-point Add functional unit performs addition or subtraction of 64-bit operands in floating-point format and executes instructions 062, 063, and 170 through 173. A result is normalized even when operands are unnormalized. (Normalized floating-point numbers are described in the subsection on floating-point arithmetic.) Out-of-range exponents are detected as described in the subsection on floating-point arithmetic.

Floating-point Add functional unit time is 6 CPs.

Floating-point Multiply functional unit

The Floating-point Multiply functional 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 can be rounded or not rounded.

Input operands are assumed to be normalized. The Floating-point Multiply functional unit delivers a normalized result only if both input operands are normalized.

Out-of-range exponents are detected as described in the subsection on floating-point arithmetic. However, if both operands have zero exponents, the result is considered as an integer product, is not normalized, and is not considered out-of-range. This case provides a fast method of computing a 48-bit integer product, although the operands in this case must be shifted before the multiply operation.

Floating-point Multiply functional unit time is 7 CPs.

Reciprocal Approximation functional unit

The Reciprocal Approximation functional unit finds the approximate reciprocal of a 64-bit operand in floating-point format. The unit executes instructions 070 and 174ij0. Since the Vector Population/Parity functional unit shares some logic with this unit, the k designator must be 0 for the reciprocal approximation instruction to be recognized.

The input operand is assumed to be normalized and if so the result is correct. The high-order bit of the coefficient is not tested but is assumed to be a 1. Out-of-range exponents are detected as described under Floating-point Arithmetic.

The Reciprocal Approximation functional unit time is 14 CPs.

ARITHMETIC OPERATIONS

Functional units in a CPU perform either twos complement integer arithmetic or floating-point arithmetic.

INTEGER ARITHMETIC

All integer arithmetic, whether 24 bits or 64 bits, is twos complement and is represented in the registers as illustrated in figure 4-4. The Address Add and Address Multiply functional units perform 24-bit arithmetic. The Scalar Add and the Vector Add functional units perform 64-bit arithmetic.

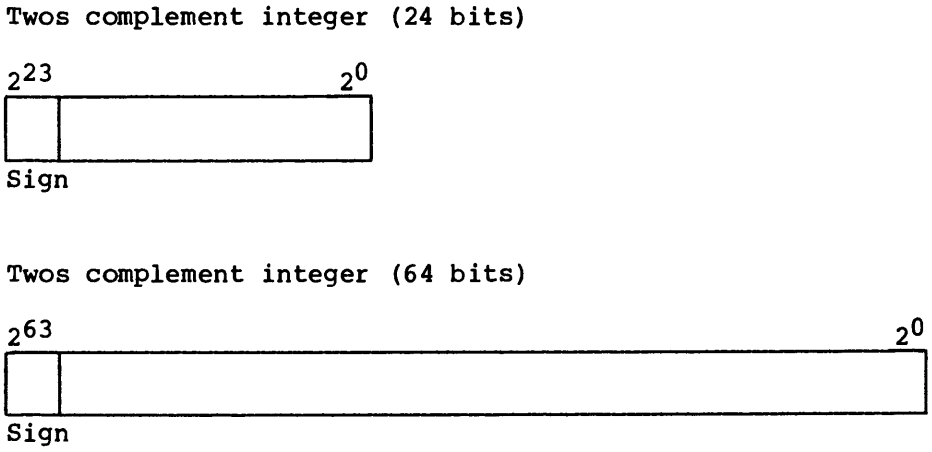


Figure 4-4. Integer data formats

Multiplication of two scalar (64-bit) integer operands is accomplished by using the floating-point multiply instruction and one of the two methods that follows. The method used depends on the magnitude of the operands and the number of bits to contain the product.

If the operands are nonzero only in the 24 least significant bits, the two integer operands can be multiplied by shifting them each left 24 bits before the multiply operation. (The Floating-point Multiply functional unit recognizes the conditions where both operands have zero exponents as a special case.) The Floating-point Multiply functional unit returns the high-order 48 bits of the product of the coefficients as the coefficient of the result and leaves the exponent field zero. See figure 4-6. If the operand coefficients are generated by other than shifting so the low-order 24 bits would be nonzero, the low-order 48 bits of the product could have been nonzero, and the high-order 48 bits (the return part) could be one larger than expected as a truncation compensation constant is always added during a multiply.

If the operands are greater than 24 bits, multiplication is done by forming multiple partial products and then shifting and adding the partial products.

Division is done by algorithm; the particular algorithm used depends on the number of bits in the quotient. The quickest and most frequently used method is to convert the numbers to floating-point format and then use the floating-point functional 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 (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 4-5. Since the coefficient is signed magnitude, it is not complemented for negative values.

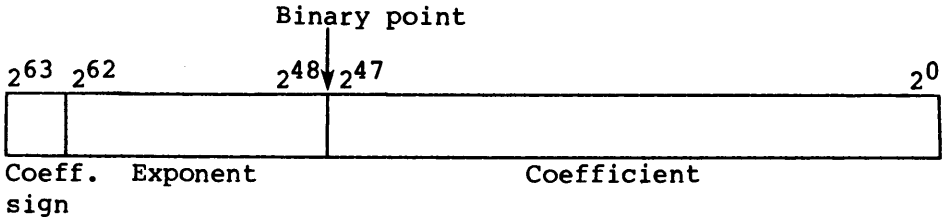


Figure 4-5. Floating-point data format

The exponent portion of the floating-point format is represented as a biased integer in bits 2^{62} through 2^{48}. 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 (note the negative range is one larger):

2^{-20000_8} through 2^{+17777_8}

In terms of decimal values, the floating-point format of the CRAY X-MP allows the accurate expression of numbers 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 0 is not generated by any floating-point functional unit, except in the case where a negative 0 is one operand going into the Floating-point Multiply functional unit.

Normalized floating-point numbers, floating-point range errors, double-precision numbers, and the addition, multiplication, and division algorithms are described in the remainder of this subsection.

Normalized floating-point numbers

A nonzero floating-point number is normalized if the most significant bit of the coefficient is nonzero. This condition implies the coefficient has been shifted as far left as possible and the exponent adjusted accordingly. Therefore, the floating-point number has no leading zeros in the coefficient. The exception is that a normalized floating-point zero is all zeros.

When a floating-point number is created by inserting an exponent of 40060_8 into a 48-bit integer word, the result should be normalized before being used in a floating-point operation. Normalization is accomplished by adding the unnormalized floating-point operand to 0. Since S0 provides a 64-bit zero when used in the Sj field of an instruction, an operand in Sk is normalized using the 062i0k instruction. Si, which can be Sk, contains the normalized result.

The 170i0k instruction normalizes Vk into Vi.

Floating-point range errors

Overflow of the floating-point range is indicated by an exponent value of 60000_8 or greater in packed format. Detection of the overflow condition initiates an interrupt if the Floating-point Mode flag is set in the Mode register and monitor mode is not in effect. The Floating-point Mode flag can be set or cleared by a user mode program.

The Cray Operating System (COS) keeps a bit in a table to indicate the condition of the mode bit. System software manipulates the mode bit and uses the table bit to indicate how the mode should be left for the user. Therefore, the user usually needs to put the appropriate bit in the table if the user changes the mode.

Floating-point range error conditions are detected by the floating-point functional units as described in the following paragraphs.

Floating-point Add functional unit - 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. This condition sets the

Floating-point Error flag with an exponent of 60000_8 being sent to the result register along with the computed coefficient, as in the following example:

```
 60000.4xxxxxxxxxxxxxx   Range error
+57777.4xxxxxxxxxxxxxx
 60000.6xxxxxxxxxxxxxx   Result register
```

NOTE

If the result of an add or subtract operation is less than the machine minimum, the error is suppressed (even though both operands have exponents greater than or equal to 60000_8) because the machine minimum takes precedence in error detection.

Floating-point Multiply functional unit - Out-of-range conditions are tested before normalizing. In the Floating-point Multiply functional unit, if the exponent of either operand is greater than or equal to 60000_8 or if the biased sum minus 1 of the two unbiased 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.

NOTE

If either operand is less than the machine minimum, the error is suppressed (even though the other operand can be out of range) because the operand that is less than the machine minimum takes precedence in error detection.

If both incoming exponents are equal to 0, the operation is treated as an integer multiply. The result is treated normally with no normalization shift of the result allowed. The result is a 48-bit quantity starting with bit 2^{47}. When using this feature, the operands should be considered as 24-bit integers in bits 2^{47} through 2^{24}. In figure 4-6, operand 1 is 4 and operand 2 is 6, producing a 48-bit result of 30_8. Bit 2^{63} obeys the usual rules for multiplying signs and the result is a sign and magnitude integer. Note the form of integers (see figure 4-4) accepted by the integer add and subtract and expected by the software is twos complement not sign and magnitude. Therefore, negative products must be converted.

If bits 2^0 through 2^{23} in operands 1 and 2 of figure 4-6 have any 1 bits, the product might be one too large (2^0) because a truncation compensation constant is added during the multiply process. (The following paragraphs discuss the truncation constant and its use.) The size of the shaded area in operands 1 and 2 (figure 4-6) does not need to be the same for both operands. To get a correct product, the only requirement is that the sum of the number of bits in the shaded area is 48 bits or more. If the sum is more than 48 bits, the binary point in the product is the number of places to the left that the sum is in excess of 48 (that is, assuming the operand binary points are at the left boundary of the shaded areas).

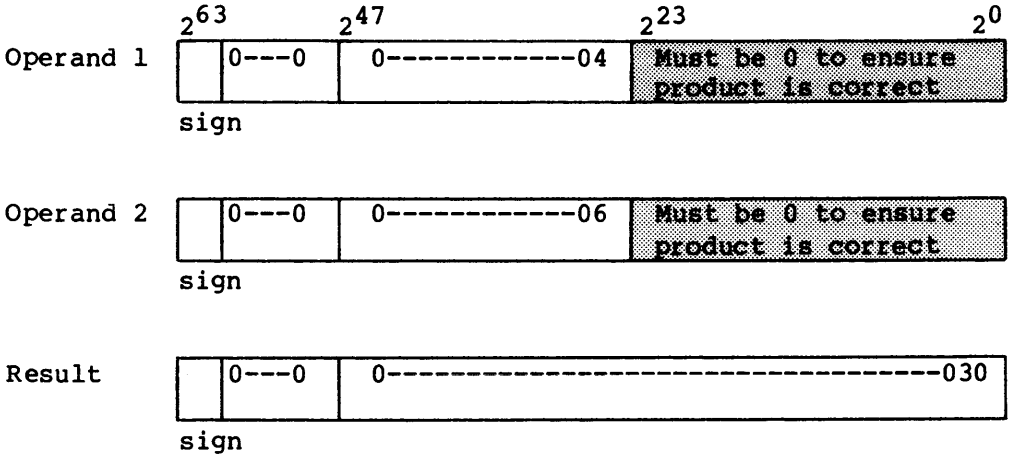


Figure 4-6. Integer multiply in Floating-point Multiply functional unit

Floating-point Reciprocal Approximation functional unit - For the Floating-point Reciprocal Approximation functional 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 and the computed coefficient are sent to the result register.

Double-precision numbers

The CPU 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, Inc.

Addition algorithm

Floating-point addition or subtraction is performed in a 49-bit register (figure 4-7). Trial subtraction of the exponents selects the operand to be shifted down for aligning the operands. The larger exponent operand carries the sign. The coefficient of the number with the smaller exponent is shifted right to align with the coefficient of the number with larger exponent. Bits shifted out of the register are lost; no round-up takes place. If the sum carries into the high-order bit, the low-order bit is discarded and an appropriate exponent adjustment is made. All results are normalized and if the result is less than the machine minimum, the error is suppressed.

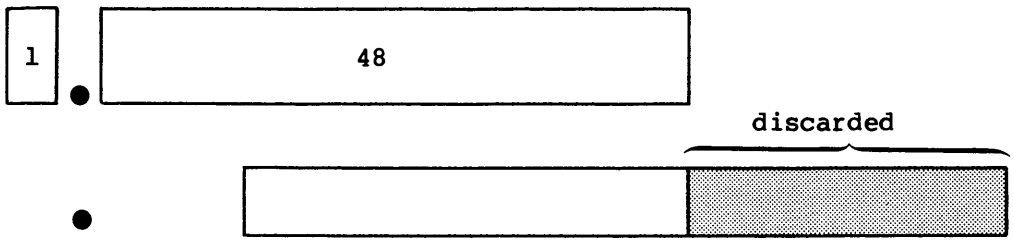


Figure 4-7. 49-bit floating-point addition

The Floating-point Add functional unit normalizes any floating-point number within the format of the CRAY X-MP floating-point number system. The functional unit right shifts 1 or left shifts up to 48 per result to normalize the result.

One zero operand and one valid operand can be sent to the Floating-point Add functional unit, and the valid operand is sent through the unit normalized. Concurrently, the functional unit checks for overflow and/or underflow; underflow results are not flagged as errors.

Multiplication algorithm

The Floating-point Multiply functional unit has the two 48-bit coefficients as input into a multiply pyramid (see figure 4-8). If the coefficients are both normalized, then a full product is either 95 bits or 96 bits, depending on the value of the coefficients. A 96-bit product is normalized as generated. A 95-bit product requires a left shift of one to generate the final coefficient. If the shift is done, the final exponent is reduced by one to reflect the shift. The following discussion and the power of two designators used assumes that the product generated is in its final form; that is, no shift was required. On the CRAY X-MP, the pyramid truncates part of the low-order bits of the 96-bit product. To adjust for this truncation, a constant is unconditionally

added above the truncation. The average value of this truncation is 9.25×2^{-56}, which was determined by adding all carries produced by all possible combinations that could be truncated and dividing the sum by the number of possible combinations. Nine carries are injected at the 2^{-56} position to compensate for the truncated bits. The effect of the truncation without compensation is at most a result coefficient one smaller than expected. With compensation, the results range from one too large to one too small in the 2^{-48} bit position with approximately 99 percent of the values having zero deviation from what would have been generated had a full 96-bit pyramid been present. The multiplication is commutative; that is, A times B equals B times A.

Rounding is optional where truncation compensation is not. The rounding method used adds a constant so that it is 50 percent high ($.25 \times 2^{-48}$; high) 38 percent of the time and 25 percent low ($.125 \times 2^{-48}$; low) 62 percent of the time resulting in near zero average rounding error. 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 the 29 most significant bits of the normalized result are transmitted back.

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

$$-0.23 \times 2^{-48} \text{ to } +0.57 \times 2^{-48}$$

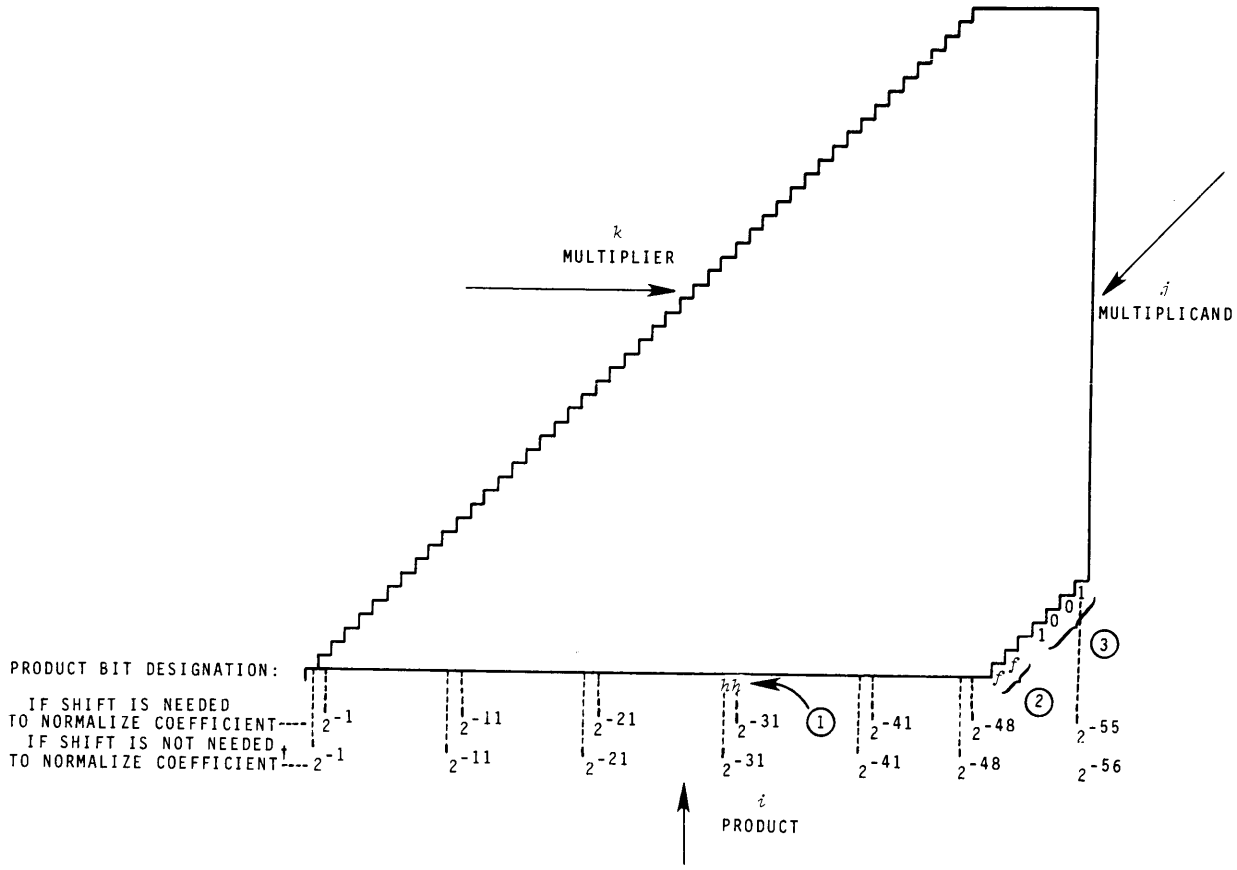
or $$-8.17 \times 10^{-16} \text{ to } +20.25 \times 10^{-16}.$$

With a full 96-bit pyramid and rounding equal to one-half the least significant bit, the variation would be expected to be:

$$-0.5 \times 2^{-48} \text{ to } +0.5 \times 2^{-48}$$

Division algorithm

The CRAY X-MP performs floating-point division through reciprocal approximation, facilitating hardware implementation of a fully segmented functional unit. Because of this segmentation, operands enter the reciprocal unit during each CP. In vector mode, results are produced at a 1-CP rate and are used in other vector operations during chaining because all functional units in the CRAY X-MP have the same result rate. The reciprocal approximation is based on Newton's method.



(1) hh = 11_2 for half-precision round, 00_2 for full-precision rounded or full-precision unrounded multiply

(2) ff = 11_2 for full-precision round, 00_2 for half-precision rounded or full-precision unrounded multiply

(3) Truncation compensation constant, 1001_2 used for all multiplies

Figure 4-8. Floating-point multiply partial-product sums pyramid

† Bit designations are used in the explanation of the Floating-point Multiply functional unit operation.

Newton's method - The division algorithm is an application of Newton's method for approximating the real roots of an arbitrary equation $F(x) = 0$, for which $F(x)$ must be twice differentiable with a continuous second derivative. The method requires making an initial approximation (guess), x_0, sufficiently close to the true root, x_t, being sought (see figure 4-9). For a better approximation, a tangent line is drawn to the graph of $y = F(x)$ at the point $(x_0, F(x_0))$. The X intercept of this tangent line is the better approximation x_1. This can be repeated using x_1 to find x_2, etc.

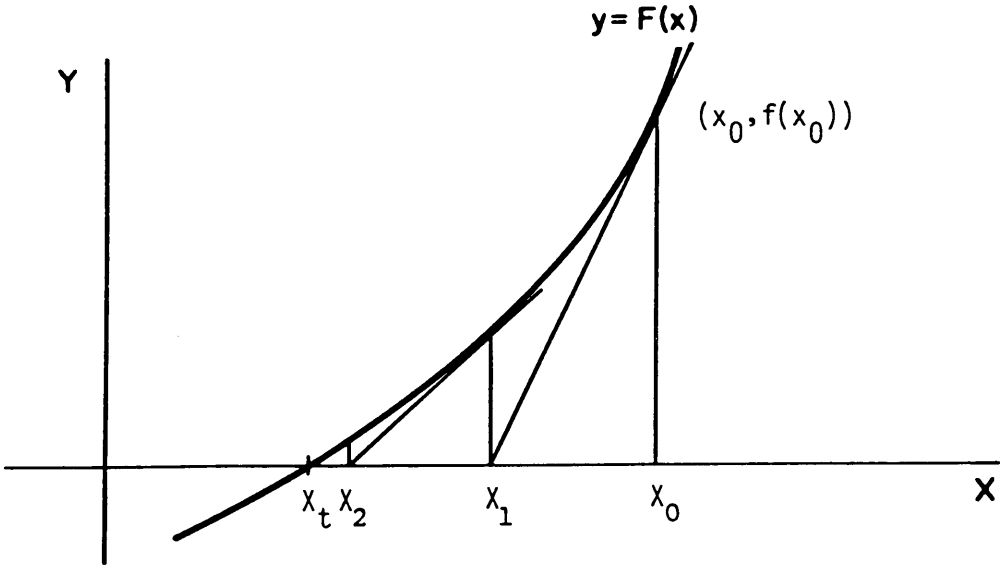


Figure 4-9. Newton's method

Derivation of the division algorithm

A definition for the derivative $F'(x)$ of a function $F(x)$ at point x_t is

$$F'(x_t) = \lim_{x \to x_t} \frac{F(x) - F(x_t)}{x - x_t}$$

if this limit exists. If the limit does not exist, $F(x)$ is not differentiable at the point t.

For any point x_i near to x_t,

$$F'(x_t) \approx \frac{F(x_i) - F(x_t)}{x_i - x_t}$$ where $\approx$ means "approximately equal to".

This approximation improves as x_i approaches x_t. Let x_i stand for an approximate solution and let x_t stand for the true answer being sought. The exact answer is then the value of x that makes F(x) equal 0. This is the case when $x=x_t$, therefore $F(x_t)$ in the equation above can be replaced by 0, giving the following approximation:

$$F'(x_t) \approx \frac{F(x_i)}{x_i - x_t} \qquad \text{Approximation (1)}$$

Notice that $x_t - x_i$ is the correction applied to an approximate answer, x_i, to give the right answer since $x_i + (x_t - x_i)$ equals x_t. Solving approximation (1) for $(x_t - x_i)$ gives:

$$x_t - x_i = \text{correction} \approx - \frac{F(x_i)}{F'(x_t)},$$

that is, $- \frac{F(x_i)}{F'(x_t)}$ is the approximate correction.

If this quantity is substituted into the approximation, then:

$$x_t \approx (x_i + \text{approximate correction}) = x_{i+1}.$$

This gives, the following equation:

$$x_{i+1} = x_i - \frac{F(x_i)}{F'(x_i)}, \qquad \text{Equation (1)}$$

where x_{i+1} is a better approximation than x_i to the true value, x_t, being sought. The exact answer is generally not obtained at once because the correction term is not generally exact. However, the operation is repeated until the answer becomes sufficiently close for practical use.

To make use of Newton's method to find the reciprocal of a number B, simply use $F(x) = (1/x - B)$.

First calculating F'(x):

where $F'(x) = \left(\frac{1}{x} - B\right)' = \left(\frac{-1}{x^2}\right)$. thus for any point $x_1 \neq 0$,

$F'(x_1) = - \frac{1}{x_1^2}$. Choosing for x, a value near $\frac{1}{B}$

and applying equation (1),

$$x_2 = x_1 - \frac{\frac{1}{x_1} - B}{-\frac{1}{x_1^2}},$$

$$x_2 = x_1 + x_1^2 \left(\frac{1}{x_1} - B\right),$$

$$x_2 = x_1 + x_1 - x_1^2 B,$$

$$x_2 = 2x_1 - x_1^2 B = x_1(2 - x_1 B).$$

On the CRAY X-MP, x_1 times the quantity in parentheses is performed by a floating-point multiply. $2-x_1B$ is performed by the reciprocal approximation instruction. x_1 is the x near 1/B and is formed by the half-precision reciprocal approximation instruction.

This approximation technique using Newton's method is implemented in the CRAY X-MP. A hardware table look up provides an initial guess, x_0, to start the process.

$x_0(2 - x_0B)$	1st approximation, I1	Done in reciprocal unit
$x_1(2 - x_1B)$	2nd approximation, I2	
$x_2(2 - x_2B)$	3rd approximation, I3	
$x_3(2 - x_3B)$	4th approximation	Done with software

The CRAY X-MP Reciprocal Approximation functional unit performs three iterations: I1, I2 and I3. I1 is accurate to 8 bits and is found after a table look-up to choose the initial guess, x_0. I2 is the second iteration and is accurate to 16 bits. I3 is the final (third) iteration answer of the Reciprocal Approximation functional unit, and its result is accurate to 30 bits.

A fourth iteration uses a special instruction within the Floating-point Multiply functional unit to calculate the correction term. This iteration is used to increase accuracy of the reciprocal unit's answer to full precision. A fifth iteration should not be done.

The division algorithm that computes S1/S2 to full-precision requires the following operations:

S3 = 1/S2	Performed by the Reciprocal Approximation functional unit
S4 = (2 - (S3 * S2))	Performed by the Floating-point Multiply functional unit in iteration mode

S5 = S4 * S3	Performed by the Floating-point Multiply functional unit using full-precision. S5 now equals 1/S2 to 48-bit accuracy.
S6 = S5 * S1	Performed by the Floating-point Multiply functional unit using full-precision rounded

The reciprocal approximation at step 1 is correct to 30 bits. An additional Newton iteration (fourth iteration) at operations 2 and 3 increases this accuracy to 48 bits. This iteration answer is applied as an operand in a full-precision rounded multiply operation to obtain the quotient accurate to 48 bits. Additional iterations should not be attempted since erroneous results are possible.

****

CAUTION

The reciprocal iteration is designed for use once with each half-precision reciprocal generated. If the fourth iteration (the programmed iteration) results in an exact reciprocal or if an exact reciprocal is generated by some other method, performing another iteration results in an incorrect final reciprocal.

****

Where 29 bits of accuracy are sufficient, the reciprocal approximation instruction is used with the half-precision multiply to produce a half-precision quotient in only two operations.

S3 = 1/S2	Performed by the Reciprocal Approximation functional unit
S6 = S1 * S3	Performed by the Floating-point Multiply functional unit in half-precision

The 19 low-order bits of the half-precision results are returned as zeros with a rounding applied to the low-order bit of the 29-bit result.

Another method of computing divisions is as follows:

S3 = 1/S2	Performed by the Reciprocal Approximation functional unit
S5 = S1 * S3	Performed by the Floating-point Multiply functional unit

S4 = (2 - (S3 * S2))	Performed by the Floating-point Multiply functional unit
S6 = S4 * S5	Performed by the Floating-point Multiply functional unit

A scalar quotient is computed in 29 CPs since operations 2 and 3 issue in successive CPs. With this method the correction to reach a full-precision reciprocal is applied after the numerator is multiplied times the half-precision reciprocal rather than before.

A vector quotient using this procedure requires less than four vector times since operations 1 and 2 are chained together. This overlaps one of the multiply operations. (A vector time is 1 CP for each element in the vector.)

****

CAUTION

The coefficient of the reciprocal produced by the alternate method can be as much as 2×2^{-48} different from the first method described for generating full-precision reciprocals. This difference can occur because one method can round up as much as twice while the other method may not round at all. One round can occur while the correction is generated and the second round can occur when producing the final quotient.

Therefore, if the reciprocals are to be compared, the same method should be used each time the reciprocals are generated. Cray FORTRAN (CFT) used a consistent method and ensures the reciprocals of numbers are always the same.

****

For example, two 64-element vectors are divided in 3 * 64 CPs plus overhead. (The overhead associated with the functional units for this case is 38 CPs).

LOGICAL OPERATIONS

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 1  1 0 1 0
Operand 2  1 1 0 0
Result     1 0 0 0
```

An operation similar to the AND function produces the following results:

```
Operand 1  1 0 1 0
Operand 2  1 1 0 0
Result     0 1 0 0
```

The logical product (AND) operation is used for masking operations where the ones specify the bits to be saved. In this variant of the AND function, the zeros specify the bits to be saved (Operand 1 is the mask).

A logical sum is the inclusive OR function:

```
Operand 1  1 0 1 0
Operand 2  1 1 0 0
Result     1 1 1 0
```

A logical difference is the exclusive OR function:

```
Operand 1  1 0 1 0
Operand 2  1 1 0 0
Result     0 1 1 0
```

A logical equivalence is the exclusive NOR function:

```
Operand 1  1 0 1 0
Operand 2  1 1 0 0
Result     1 0 0 1
```

The merge uses two operands and a mask to produce results as follows:

```
Operand 1  1 0 1 0 1 0 1 0
Operand 2  1 1 0 0 1 1 0 0
Mask       1 1 1 1 0 0 0 0
Result     1 0 1 0 1 1 0 0
```

The bits of operand 1 pass where the mask bit is 1. The bits of operand 2 pass where the mask bit is 0.

CPU INSTRUCTIONS 5

INSTRUCTION FORMAT

Each instruction used in a CRAY X-MP computer is either a 1-parcel (16-bit) instruction or a 2-parcel (32-bit) instruction. Instructions are packed four parcels per word. Parcels in a word are numbered 0 through 3 from left to right and any parcel position can be addressed in branch instructions. A 2-parcel instruction begins in any parcel of a word and can span a word boundary. For example, a 2-parcel instruction beginning in the fourth parcel of a word ends in the first parcel of the next word. No padding to word boundaries is required. Figure 5-1 illustrates the general form of instructions.

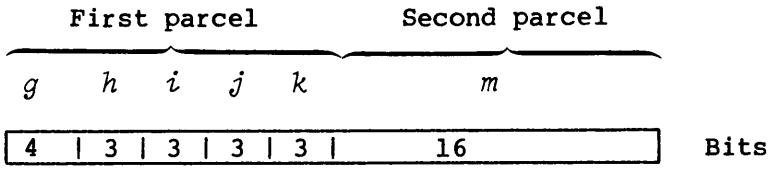


Figure 5-1. General form for instructions

Four variations of this general format use the fields differently; two forms are 1-parcel formats and two are 2-parcel formats. The formats of these four variations are described below.

1-PARCEL INSTRUCTION FORMAT WITH DISCRETE *j* AND *k* FIELDS

The most common of the 1-parcel instruction formats uses the *i*, *j*, and *k* fields as individual designators for operand and result registers (see figure 5-2). The *g* and *h* fields define the operation code. The *i* field designates a result register and the *j* and *k* fields designate operand registers. Some instructions ignore one or more of the *i*, *j*, and *k* fields. The following types of instructions use this format.

- Arithmetic
- Logical
- Double shift
- Floating-point constant

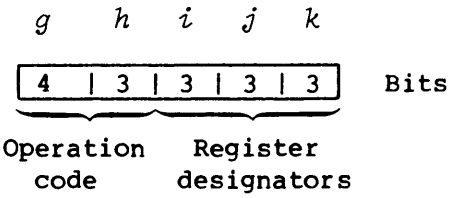


Figure 5-2. 1-parcel instruction format with discrete *j* and *k* fields

1-PARCEL INSTRUCTION FORMAT WITH COMBINED *j* AND *k* FIELDS

Some 1-parcel instructions use the *j* and *k* fields as a combined 6-bit field (see figure 5-3). The *g* and *h* fields contain the operation code, and the *i* field is generally a destination register identifier. The combined *j* and *k* fields generally contain a constant or a B or T register designator. The branch instruction 005 and the following types of instructions use the 1-parcel instruction format with combined *j* and *k* fields.

- Constant
- B and T register block memory transfer
- B and T register data transfer
- Single shift
- Mask

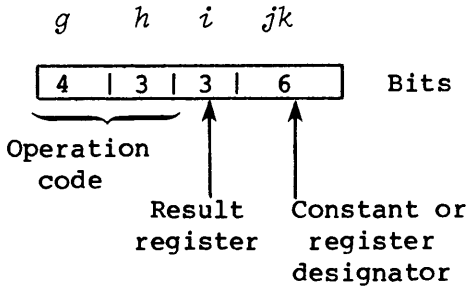


Figure 5-3. 1-parcel instruction format with combined *j* and *k* fields

2-PARCEL INSTRUCTION FORMAT WITH COMBINED *j*, *k*, AND *m* FIELDS

The instruction type for a 22-bit immediate constant uses the combined *j*, *k*, and *m* fields to hold the constant. The 7-bit *gh* field contains an operation code, and the 3-bit *i* field designates a result register. The instruction type using this format transfers the 22-bit *jkm* constant to an A or S register.

The instruction type used for scalar memory transfers also requires a 22-bit *jkm* field for an address displacement. This instruction type uses the 4-bit *g* field for an operation code, the 3-bit *h* field to designate an address index register, and the 3-bit *i* field to designate a source or result register. (See subsection on special register values.)

Figure 5-4 shows the two general applications for the 2-parcel instruction format with combined *j*, *k*, and *m* fields.

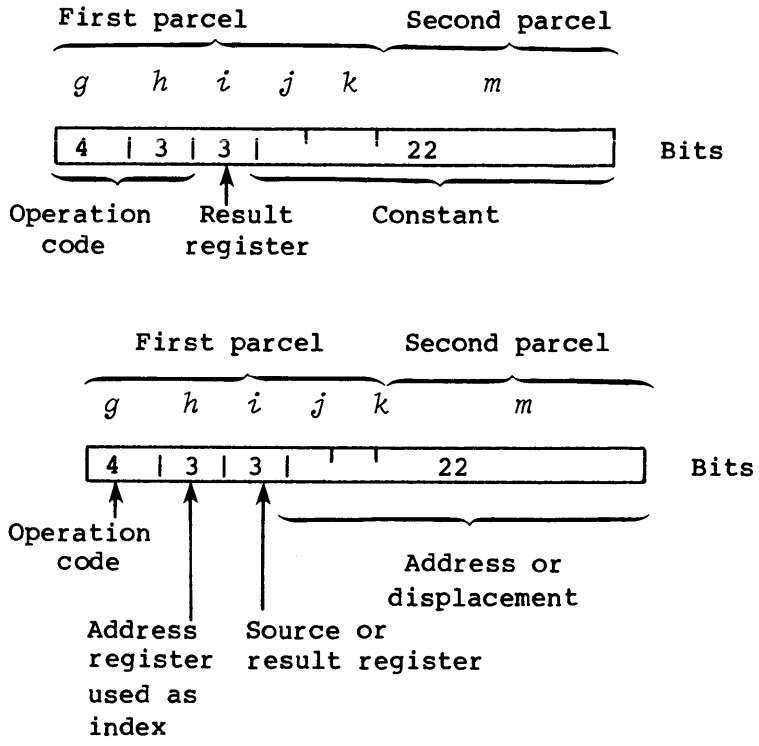


Figure 5-4. 2-parcel instruction format with combined *j*, *k*, and *m* fields

2-PARCEL INSTRUCTION FORMAT WITH COMBINED *i*, *j*, *k*, AND *m* FIELDS

The 2-parcel branch instruction type uses the combined *i*, *j*, *k*, and *m* fields to contain a 24-bit address that allows branching to an instruction parcel (see figure 5-5). A 7-bit operation code (*gh*) is followed by an *ijkm* field. The high-order bit of the *i* field is unused.

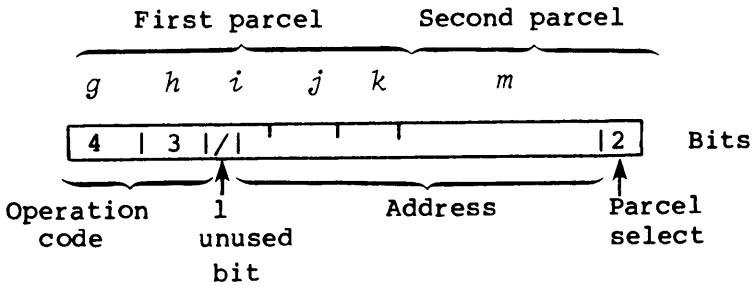


Figure 5-5. 2-parcel instruction format with combined *i*, *j*, *k*, and *m* fields

SPECIAL REGISTER VALUES

If the S0 and A0 registers are referenced in the *j* or *k* fields of an instruction, the contents of the respective register are not used; instead, a special operand is generated. The special value is available regardless of existing A0 or S0 reservations (and in this case are not checked). This use does not alter the actual value of the S0 or A0 register. If S0 or A0 is used in the *i* field as the operand, the actual value of the register is provided. The table below shows the special register values.

Field	Operand value
A*h*, *h*=0	0
A*i*, *i*=0	(A0)
A*j*, *j*=0	0
A*k*, *k*=0	1
S*i*, *i*=0	(S0)
S*j*, *j*=0	0
S*k*, *k*=0	2^{63}

INSTRUCTION ISSUE

Instructions are read one parcel at a time from the instruction buffers and delivered to the Next Instruction Parcel (NIP) register. The instruction is then passed to the Current Instruction Parcel (CIP) register when the previous instruction issues. An instruction in the CIP register issues when conditions in the functional unit and registers are such that functions required for execution can be performed without conflicting with a previously issued instruction. Instruction parcels can issue out of the CIP register at a maximum rate of one per clock period.

Execution times (the time from issue to delivery of data to the destination operating registers) are fixed for instructions 000 through 077, except those that reference memory (instructions 000, 004, branch instructions 005 through 017, and block transfer instructions 034 through 037). Scalar memory instructions 100 through 137 complete in variable lengths of time. Vector operation instructions 140 through 177 complete in a fixed time if the instructions are not chained to memory fetches.

Execution times can be affected by instruction 0034*jk*, which tests and sets the semaphore designated by *jk*. If the semaphore is set, instruction issue is held until the other CPU clears that semaphore. If the semaphore is clear, the instruction issues and sets the semaphore. If all CPUs in a cluster are holding issue on a test and set, a flag is set in the Exchange Package (if not in monitor mode) and an exchange occurs. If an interrupt occurs while a test and set instruction is holding in the CIP register, a flag is set in the Exchange Package, CIP and NIP registers clear, and an exchange occurs with the P register pointing to the test and set instruction.

Entry to the NIP register is blocked for the second parcel of a 2-parcel instruction, leaving NIP blanked. Instead, the parcel is delivered to the Lower Instruction Parcel (LIP) register. The zeros in NIP (the pseudo second parcel) are transferred to CIP and issued as a do-nothing instruction.

When special register values (A0 or S0) are selected by an instruction for A*h*, A*j*, A*k*, S*j*, or S*k*, the normal "hold issue until operand ready" conditions do not apply. These values are always immediately available.

INSTRUCTION DESCRIPTIONS

This section contains detailed information about individual instructions or groups of related instructions. Each instruction begins with boxed information consisting of the Cray Assembly Language (CAL) syntax

format, a brief description of each instruction, and the octal code sequence defined by the *gh* fields. The appearance of an *m* in a format designates an instruction consisting of two parcels.

Following the boxed 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 delaying issue of an instruction until conditions are met.

Instruction issue time assumes that if an instruction issues at clock period *n* (CP *n*), the next instruction issues at CP *n* + issue time[†] if its own issue conditions have been met.

The following special characters can appear in the operand field description of symbolic machine instructions and are used by the assembler in determining the operation to be performed.

+	Arithmetic sum of adjoining registers
-	Arithmetic difference of adjoining registers
*	Arithmetic product of adjoining registers
/	Division or reciprocal
#	Use ones complement
>	Shift value or form mask from left to right
<	Shift value or form mask from right to left
&	Logical product of adjoining registers
!	Logical sum of adjoining registers
\	Logical difference of adjoining registers

In some instructions, register designators are prefixed by the following letters, which have special meaning to the assembler.

F	Floating-point operation
H	Half-precision operation
R	Rounded operation
I	Reciprocal iteration
P	Population count
Q	Population count parity
Z	Leading zero count

---

† Previous instruction issued

INSTRUCTION 000

CAL Syntax	Description	Octal Code
ERR	Error exit	000000

Instruction 000 is treated as an error condition and an exchange sequence occurs. Content of the instruction buffers is voided by the exchange sequence. Instruction 000 halts execution of an incorrectly coded program branching into an unused area of memory (if memory was backgrounded with zeros) or into a data area (if the data is positive integers, right-justified ASCII, or floating-point zero). If monitor mode is not in effect, the Error Exit flag in the F register is set. All instructions issued before this instruction are run to completion. When results of previously issued instructions arrive at the operating registers, an exchange occurs to the Exchange Package designated by contents of the XA register. The program address stored during the exchange on the terminating exchange sequence is the contents of the P register advanced by one count (that is, the address of the instruction following the error exit instruction).

HOLD ISSUE CONDITIONS: Any A, S, or V register reserved

EXECUTION TIME: Instruction issue, 40 CPs; this time includes an exchange sequence (24 CPs) and a fetch operation (16 CPs).

SPECIAL CASES: None

CAL Syntax	Description	Octal Code
CA,A*j* A*k*	Set the Current Address (CA) register for the channel indicated by (A*j*) to (A*k*) and activate the channel	0010*jk*
CL,A*j* A*k*	Set the Limit Address (CL) register for the channel indicated by (A*j*) to (A*k*)	0011*jk*
CI,A*j*	Clear the interrupt flag and error flag for the channel indicated by (A*j*); clear device master-clear (output channel)	0012*j*0
MC,A*j*	Clear the interrupt flag and error flag for the channel indicated by (A*j*); set device master-clear (output channel); clear device ready-held (input channel)	0012*j*1
XA A*j*	Enter the XA register with (A*j*)	0013*j*0

Instructions 0010 through 0013 are privileged to monitor mode and provide operations useful to the operating system. Functions are selected through the *i* designator. Instructions are treated as pass instructions if the monitor mode bit is not set.

When the *i* designator is 0, 1, or 2, the instruction controls operation of the I/O channels. Each channel has two registers directing 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 first and then CA sets, activating the channel. As transfer continues, CA is incremented toward CL. When (CA) is equal to (CL), transfer is complete for words at initial (CA) through (CL)-1. When the *j* designator is 0 or when the 4 low-order bits of A*j* are less than 10_8, the functions are executed as pass instructions. Valid channel numbers are $10\text{-}17_8$. 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 (A*j*) to the XA register. When the *j* designator is 0, the XA register is cleared.

Instruction 012*j*0 is used to clear the device Master Clear. For instruction 0012, if the *k* designator is 1 for an output channel, the master clear is set; if the *k* designator is 1 for an input channel, the ready flag is cleared.

HOLD ISSUE CONDITIONS: For instructions 0010 and 0011, A*j* or A*k* reserved (except A0)

For instructions 0012 or 0013, A*j* reserved (except A0)

EXECUTION TIME: Instruction issue, 1 CP

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

For instructions 0010, 0011, and 0012:
If *j*=0, the instruction is a no-op.
If 4 low-order bits of (A*j*) are less than 10_8, the instruction is a no-op, (that is, 20 through 27 are invalid, 30 through 37 are valid, 40 through 47 are invalid, 50 through 57 are valid, etc.).
If *k*=0, CA or CL is set to 1.

For instruction 0012:
The correct priority interrupting channel number cannot be read (via instruction 033) until 2 CPs after issue of instruction 0012.

For instruction 0013:
If *j*=0, XA register is cleared.

NOTE

Because there is no hardware interlock between the two CPUs, it is possible to have two CPUs issuing these instructions at the same time; however, undetermined results can occur.

Software must ensure only one CPU is servicing I/O at a time while in monitor mode.

INSTRUCTION 0014

CAL Syntax	Description	Octal Code
RT Sj	Enter the Real-time Clock register with (Sj)	0014j0
IP 1	Set interprocessor interrupt request of other processor	001401
IP 0	Clear received interprocessor interrupt request from other processor	001402
CLN 0	Cluster number = 0	001403
CLN 1	Cluster number = 1	001413
CLN 2	Cluster number = 2	001423
CLN 3	Cluster number = 3	001433
PCI Sj	Enter Interrupt Interval (II) register with (Sj)	0014j4
CCI	Clear the programmable clock interrupt request	001405
ECI	Enable programmable clock interrupt request	001406
DCI	Disable programmable clock interrupt request	001407

Instruction 0014 performs specialized functions for managing the real-time and programmable clocks and handles interprocessor interrupt requests and cluster number operations. Instruction 0014 is privileged to monitor mode and is treated as a pass instruction if the monitor mode bit is not set.

When the *k* designator is 0, the instruction loads the contents of the *Sj* register into the RTC register. When the *j* designator is 0 or (*Sj*)=0, the RTC register is cleared.

When the *k* designator is 1, the instruction sets the internal CPU interrupt request in the other CPU. If the other CPU is not in monitor mode, the ICP (Interrupt from Internal CPU) flag sets in the F register causing an interrupt. The request remains until cleared by the receiving CPU issuing instruction 001402.

When the *k* designator is 2, the instruction clears the internal CPU interrupt request set by the other CPU.

When the *k* designator is 3, the instruction sets the cluster number to *j* to make the following cluster selections:

CLN = 0 No cluster; all shared register and semaphore operations are no-ops, (except SB, ST, or SM register reads, which return a 0 value to A*i* or S*i*).

CLN = 1 Cluster 1

CLN = 2 Cluster 2

CLN = 3 Cluster 3

Clusters 1, 2, and 3 each have a separate set of SM, SB, and ST registers.

When the *k* designator is 4, the instruction loads the low-order 32 bits from the S*j* register into both the II register and the ICD counter. When the *j* designator is 0 or (S*j*)=0, II and ICD are cleared.

When the *k* designator is 5, the instruction clears the programmable clock interrupt request if the request is previously set by ICD counting down to 0.

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

When the *k* designator is 7, the instruction disables repeated programmable clock interrupt requests until an instruction 001406 is executed to enable the requests.

HOLD ISSUE CONDITIONS: S*j* reserved (except S0)

For instruction 0014*j*3, hold issue 2 CPs

EXECUTION TIME: Instruction issue, 1 CP

SPECIAL CASES: If the program is not in monitor mode, these instructions become no-ops but all hold issue conditions remain effective.

For instructions 0014*j*0 and 0014*j*4, if *j*=0, (S*j*)=0.

For instruction 0014*j*0, the value is entered into the RTC register 4 CPs after instruction 0014*j*0 issues.

CAL Syntax	Description	Octal Code
VL Ak	Transmit (Ak) to VL register	00200k
VL 1†	Transmit 1 to VL register	002000

Instruction 00200k enters the VL register with a value determined by the contents of Ak. The low-order 6 bits of (Ak) are entered into the VL register. The 7th bit of VL is set if the 6 low-order bits of (Ak)=0.

For example, if (Ak)=0 or a multiple of 100_8, then VL=100_8. The content of VL is always between 1 and 100_8.

Instruction 002000 transmits the value of 1 to the VL register.

HOLD ISSUE CONDITIONS: Ak reserved (except A0)

EXECUTION TIME: Instruction issue, 1 CP
VL register ready, 1 CP

SPECIAL CASES: Maximum vector length is 64.
(Ak)=1 if k=0.
(VL)=100_8 if k≠0 and (Ak)=0 or a multiple of 100_8.

† Special CAL syntax

CAL Syntax	Description	Octal Code
EFI	Enable interrupt on floating-point error	002100
DFI	Disable interrupt on floating-point error	002200
ERI	Enable interrupt on operand (address) range error	002300
DRI	Disable interrupt on operand (address) range error	002400
DBM	Disable bidirectional memory transfers	002500
EBM	Enable bidirectional memory transfers	002600
CMR	Complete memory references	002700

Instruction 002100 sets the Floating-point Mode flag in the M register. Instruction 002200 clears the Floating-point Mode flag in the M register. The two instructions do not check the previous state of the flag. When set, the Floating-point Mode flag enables interrupts on floating-point range errors as described in section 4. Issuing either of these instructions also clears the Floating-Point Error Status flag.

Instruction 002300 sets the Operand Range Mode flag in the M register. Instruction 002400 clears the Operand Range Mode flag in the M register. The two instructions do not check the previous state of the flag. When set, the Operand Range Mode flag enables interrupts on operand (address) range errors as described in section 3.

Instruction 002500 enables the bidirectional memory mode. Instruction 002600 disables the bidirectional memory mode. Block reads and writes can operate concurrently in bidirectional memory mode. If the bidirectional memory mode is disabled, only block reads can operate concurrently.

Instruction 002700 assures completion of all memory references within a particular CPU issuing the instruction. Instruction 002700 does not issue until all memory references before this instruction are at the stage of execution where completion occurs in a fixed amount of time. For example, a load of any data that has been stored by the CPU issuing instruction CMR, 002700 is assured of receiving the updated data if the load is issued after the CMR instruction. Synchronization of memory references between processors can be done by this instruction in conjunction with semaphore instructions.

HOLD ISSUE CONDITIONS: Instructions 002500 and 002600, hold issue 2 CPs

Instruction 002700, ports A, B, C busy

Instruction 002700, scalar memory reference active in clock period 1, 2, or 3

A*k* reserved (except A0)

EXECUTION TIME: Instruction issue, 1 CP

SPECIAL CASES: Instructions 002100 and 002200 are issued even if there are other floating-point operations in process resulting from previous issues. The interrupts are enabled or disabled at CP + 1; floating-point overflows occurring after that time cause interrupts if they are enabled even if the overflow is generated by a previously issued floating-point instruction.

Instructions 002300 and 002400 are issued even if there are other memory references in process resulting from previous issues. The interrupts are enabled or disabled at CP + 1; operand range errors occurring after that time cause interrupts if they are enabled even if the operand range error is generated by a previous memory reference.

CAL Syntax	Description	Octal Code
VM Sj	Transmit (Sj) to VM register	0030j0
VM 0†	Clear VM register	003000
SMjk 1,TS	Test and set semaphore jk, $0 \leq jk \leq 31_{10}$	0034jk
SMjk 0	Clear semaphore jk, $0 \leq jk \leq 31_{10}$	0036jk
SMjk 1	Set semaphore jk, $0 \leq jk \leq 31_{10}$	0037jk

Instruction 0030j0 enters the VM register with the contents of Sj. The VM register is cleared if the j designator is 0 in instruction 003000. These instructions are used in conjunction with the vector merge instructions (146 and 147) in which an operation is performed depending on the contents of VM.

Instruction 0034jk tests and sets the semaphore designated by jk. If the semaphore is set, issue is held until the other CPU clears that semaphore. If the semaphore is clear, the instruction issues and sets the semaphore. If all CPUs in a cluster are holding issue on a test and set, the DL flag is set in the Exchange Package (if not in monitor mode) and an exchange occurs. If an interrupt occurs while a test and set instruction is holding in the CIP register, the WS flag in the Exchange Package sets, CIP and NIP registers clear, and an exchange occurs with the P register pointing to the test and set instruction. The SM register is 32 bits with SM0 being the most significant bit.

Instruction 0036jk clears the semaphore designated by jk.

Instruction 0037jk sets the semaphore designated by jk.

HOLD ISSUE CONDITIONS: For instruction 0030j0:
Sj reserved (except S0)
Instruction 003 in process, unit busy 1 CP
Instruction 14x in process, unit busy (VL)+5 CPs
Instruction 175 in process, unit busy (VL)+5 CPs

† Special CAL syntax

HOLD ISSUE CONDITIONS: (continued)

For instruction 0034*jk*:
If current Cluster Number≠0 and SM*jk* is set, holds issue until other CPU in the same cluster clears the semaphore.

EXECUTION TIME:

Instruction issue, 1 CP

SPECIAL CASES:

(S*j*)=0 if *j*=0.

Instructions 0034*jk*, 0036*jk*, and 0037*jk* are no-ops if CLN=0.

CAL Syntax	Description	Octal Code
EX	Normal exit	004000

Instruction 004 causes an exchange sequence which voids the contents of the instruction buffers. If monitor mode is not in effect, the Normal Exit flag in the F register is set. All instructions issued before this instruction are run to completion; that is, when all results arrive at the operating registers because of previously issued instructions, an exchange sequence occurs to the Exchange Package designated by the contents of the XA register. The program address stored into the Exchange Package is advanced one count from the address of the normal exit instruction. Instruction 004 is used to issue a monitor request from a user program.

HOLD ISSUE CONDITIONS: Any A, S, or V register reserved

EXECUTION TIME: Instruction issue, 40 CPs; this time includes an exchange sequence (24 CPs) and a fetch operation (16 CPs).

SPECIAL CASES: None

INSTRUCTION 005

CAL Syntax	Description	Octal Code
J B*jk*	Branch to (B*jk*)	0050*jk*

Instruction 005 sets the P register to the 24-bit parcel address specified by the contents of B*jk* causing execution to continue at that address. The instruction is used to return from a subroutine.

HOLD ISSUE CONDITIONS: Instruction 034 or 035 in process

Instruction 025 issued in the previous CP

Second parcel in a different buffer, 2 CP delay

Second parcel not in a buffer

EXECUTION TIME: Instruction issue:
Instruction parcel and following parcel both in a buffer and branch address in a buffer, 7 CPs

Instruction parcel and following parcel both in a buffer and branch address not in a buffer, 18 CPs. Additional time is needed if a memory conflict exists. The time to resolve a memory conflict depends on factors present.

SPECIAL CASES: Instruction 0050*jk* executes as if it were a 2-parcel instruction. Even though the parcel following the first parcel of instruction 0050*jk* is not used, it can cause a delay of instruction 0050*jk* if it is out of buffer. See execution times above.

INSTRUCTION 006

CAL Syntax	Description	Octal Code
J *exp*	Branch to *ijkm*	006*ijkm*

The 2-parcel instruction 006 sets the P register to the parcel address specified by the low-order 24 bits of the *ijkm* field. Execution continues at that address. The high-order bit of the *ijkm* field is ignored.

HOLD ISSUE CONDITIONS: Second parcel in different buffer, 2 CP delay

Second parcel not in a buffer

EXECUTION TIME: Instruction issue:

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

Both parcels of instruction in the same buffer and branch address not in a buffer, 16 CPs. Additional time is needed if a memory conflict exists. The time to resolve a memory conflict depends on factors present.

SPECIAL CASES: None

INSTRUCTION 007

CAL Syntax	Description	Octal Code
R *exp*	Return jump to *ijkm*; set B00 to (P)+2.	007*ijkm*

The 2-parcel instruction 007 sets register B00 to the address of the parcel following the second parcel of the instruction. The P register is then set to the parcel address specified by the low-order 24 bits of the *ijkm* field. Execution continues at that address. The high-order bit of the *ijkm* field is ignored. This instruction provides a return linkage for subroutine calls. The subroutine is entered via a return jump. The subroutine can return to the caller at the instruction following the call by executing a branch to the contents of the B00 register.

HOLD ISSUE CONDITIONS: Instruction 034 or 035 in process

Second parcel in a different buffer, 2 CP delay

Second parcel not in a buffer

EXECUTION TIME: Instruction issue:
Both parcels of instruction in the same buffer and branch address in a buffer, 5 CPs

Both parcels of instruction in the same buffer and branch address not in a buffer, 16 CPs. Additional time is needed if a memory conflict exists. The time to resolve a memory conflict depends on factors present.

SPECIAL CASES: None

CAL Syntax		Description	Octal Code
JAZ	*exp*	Branch to *ijkm* if (A0)=0	010*ijkm*
JAN	*exp*	Branch to *ijkm* if (A0)≠0	011*ijkm*
JAP	*exp*	Branch to *ijkm* if (A0) positive, includes (A0)=0	012*ijkm*
JAM	*exp*	Branch to *ijkm* if (A0) negative	013*ijkm*

The 2-parcel instructions 010 through 013 test the contents of A0 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 24 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: A0 busy in any one of the previous 3 CPs

Second parcel in a different buffer, 2 CP delay

Second parcel not in a buffer

EXECUTION TIME: Instruction issue for branch taken:

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

Both parcels of instruction in the same buffer, branch taken, and branch address not in a buffer, 16 CPs. Additional time is needed if a memory conflict exists. The time to resolve a memory conflict is indeterminate.

Both parcels of instruction in different buffers, branch taken, and branch address in a buffer; (timing to be supplied).

Both parcels of instruction in different buffers, branch taken, and branch address not in a buffer; (timing to be supplied).

EXECUTION TIME: (continued)

Second parcel of instruction not in a buffer, branch taken, and branch address in a buffer; (timing to be supplied).

Second parcel of instruction not in a buffer, branch taken, and branch address not in buffer; (timing to be supplied).

Instruction issue for branch not taken:
Both parcels of instruction in the same buffer, branch not taken, and next instruction in the same instruction buffer, 2 CPs

Both parcels of instruction in the same buffer, branch not taken, and next instruction in different instruction buffer, 4 CPs

Both parcels of instruction in the same buffer and branch not taken with next instruction in memory; (timing to be supplied).

Both parcels of instruction in different buffers and branch not taken; (timing to be supplied).

Second parcel of instruction not in a buffer and branch not taken; (timing to be supplied).

SPECIAL CASES:

(A0)=0 is considered a positive condition.

CAL Syntax	Description	Octal Code
JSZ *exp*	Branch to *ijkm* if (S0)=0	014*ijkm*
JSN *exp*	Branch to *ijkm* if (S0)≠0	015*ijkm*
JSP *exp*	Branch to *ijkm* if (S0) positive, includes (S0)=0	016*ijkm*
JSM *exp*	Branch to *ijkm* if (S0) negative	017*ijkm*

The 2-parcel instructions 014 through 017 test the contents of S0 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 24 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: S0 busy in any one of the previous 3 CPs

Second parcel in a different buffer, 2 CP delay

Second parcel not in a buffer

EXECUTION TIME: Instruction issue for branch taken:

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

Both parcels of instruction in the same buffer, branch taken, and branch address not in a buffer, 16 CPs. Additional time is needed if a memory conflict exists. The time to resolve a memory conflict is indeterminate.

Both parcels of instruction in different buffers, branch taken, and branch address in a buffer; (timing to be supplied).

Both parcels of instruction in different buffers, branch taken, and branch address not in a buffer; (timing to be supplied).

EXECUTION TIME: (continued)

Second parcel of instruction not in a buffer, branch taken, and branch address in a buffer; (timing to be supplied).

Second parcel of instruction not in a buffer, branch taken, and branch address not in buffer; (timing to be supplied).

Instruction issue for branch not taken:
Both parcels of instruction in the same buffer, branch not taken, and next instruction in the same instruction buffer, 2 CPs

Both parcels of instruction in the same buffer, branch not taken, and next instruction in different instruction buffer, 4 CPs

Both parcels of instruction in the same buffer and branch not taken with next instruction in memory; (timing to be supplied).

Both parcels of instruction in different buffers and branch not taken; (timing to be supplied).

Second parcel of instruction not in a buffer and branch not taken; (timing to be supplied).

SPECIAL CASES:

(S0)=0 is considered a positive condition.

CAL Syntax	Description	Octal Code
A*i* *exp*	Transmit *jkm* to A*i*	020*ijkm*
A*i* *exp*	Transmit ones complement of *jkm* to A*i*	021*ijkm*

The 2-parcel instruction 020 enters a 24-bit value into A*i* composed of the 22-bit *jkm* field and 2 high-order bits of 0.

The 2-parcel instruction 021 enters a 24-bit value that is the complement of a value formed by the 22-bit *jkm* field and 2 high-order bits of 0 into A*i*. The complement is formed by changing all 1 bits to 0 and all 0 bits to 1. Thus, for instruction 021, the high-order 2 bits of A*i* are set to 1. The instruction provides a means of entering a negative value into A*i*. However, if the instruction is used to enter a negative number, the positive number used in the *jkm* field must be one smaller than the absolute value of the expected final negative number.

HOLD ISSUE CONDITIONS: A*i* reserved

Second parcel not in a buffer

EXECUTION TIME: Instruction issue:

Both parcels in same buffer, 2 CPs

Both parcels in different buffers, 4 CPs

A*i* ready, 1 CP

SPECIAL CASES: None

INSTRUCTION 022

CAL Syntax	Description	Octal Code
A*i* *exp*	Transmit *jk* to A*i*	022*ijk*

Instruction 022 enters the 6-bit quantity from the *jk* field into the low-order 6 bits of A*i*. The high-order 18 bits of A*i* are zeroed. No sign extension occurs.

HOLD ISSUE CONDITIONS: A*i* reserved

EXECUTION TIME: Instruction issue, 1 CP

A*i* ready, 1 CP

SPECIAL CASES: None

CAL Syntax	Description	Octal Code
A*i* S*j*	Transmit (S*j*) to A*i*	023*ij*0
A*i* VL	Read vector length	023*i*01

Instruction 023*ij*0 enters the low-order 24 bits of (S*j*) into A*i*. The high-order bits of (S*j*) are ignored.

Instruction 023*i*01 enters the content of the VL register into A*i*.

HOLD ISSUE CONDITIONS: A*i* reserved

For instruction 023*ij*0, S*j* reserved (except S0)

EXECUTION TIME: Instruction issue, 1 CP

A*i* ready, 1 CP

SPECIAL CASES: (S*j*)=0 if *j*=0.

If (A1)=0, the sequence:
```
VL    A1
A2    VL
```
leaves (A2)=100_8

If (A1)=23_8, the sequence:
```
VL    A1
A2    VL
```
leaves (A2)=23_8

If (A1)=123_8, the sequence:
```
VL    A1
A2    VL
```
leaves (A2)=23_8

The 2^6 bit in the VL is a 1 if the low-order 6 bits are 0; otherwise, the 2^6 bit is a 0.

CAL Syntax	Description	Octal Code
A*i* B*jk*	Transmit (B*jk*) to A*i*	024*ijk*
B*jk* A*i*	Transmit (A*i*) to B*jk*	025*ijk*

Instruction 024 enters the contents of B*jk* into A*i*.

Instruction 025 enters the contents of A*i* into B*jk*.

HOLD ISSUE CONDITIONS: Instruction 034 or 035 in process

For instruction 024*ijk*, instruction 025*ijk* issued in previous CP

A*i* reserved

EXECUTION TIME: For instruction 024, A*i* ready, 1 CP

Instruction issue, 1 CP

SPECIAL CASES: None

CAL Syntax	Description	Octal Code
A*i* PS*j*	Population count of (S*j*) to A*i*	026*ij*0
A*i* QS*j*	Population count parity of (S*j*) to A*i*	026*ij*1
A*i* SB*j*	Transfer (SB*j*) to A*i*	026*ij*7

Instruction 026*ij*0 counts the number of bits set to 1 in (S*j*) and enters the result into the low-order 7 bits of A*i*. The high-order 17 bits of A*i* are zeroed. If (S*j*)=0, then (A*i*)=0.

Instruction 026*ij*1 counts the number of bits set to 1 in (S*j*). Then, the low-order bit, showing the odd/even state of the result is transferred to the low-order bit position of the A*i* register. The high-order 23 bits are cleared. The actual population count is not transferred.

Instructions 026*ij*0 and 026*ij*1 are executed in the Population/ Leading Zero Count functional unit.

Instruction 026*ij*7 transfers the contents of the SB*j* register shared between the CPUs to A*i*.

HOLD ISSUE CONDITIONS: A*i* reserved

S*j* reserved (except S0)

For instruction 026*ij*7, shared register access conflict due to:

Instruction 026*ij*7 or 072*ij*3 in CIP of other CPU, regardless of clustering, at the same CP; or

Instruction 027*ij*7 or 073*ij*3 issued by both CPUs, regardless of clustering, at 3 CPs earlier.

EXECUTION TIME: Instruction issue, 1 CP

For instructions 026*ij*0 and 026*ij*1, A*i* ready 4 CPs

For instruction 026*ij*7, A*i* ready 1 CP

SPECIAL CASES:

For instructions 026*ij*0 and 026*ij*1, (A*i*)=0 if *j*=0.

For instruction 026*ij*7, (A*i*)=0 if CLN=0.

For instruction 026*ij*7:
If instruction 027*ij*7, write SB*j*, has just been issued within the previous 2 CPs, then the original value (instead of new value) of (SB*j*) is delivered to A*i* as a result of this instruction.

CAL Syntax	Description	Octal Code
A*i* ZS*j*	Leading zero count of (S*j*) to A*i*	027*ij*0
SB*j* A*i*	Transfer (A*i*) to SB*j*	027*ij*7

Instruction 027*ij*0 counts the number of leading zeros in S*j* and enters the result into the low-order 7 bits of A*i*. The high-order 17 bits of A*i* are zeroed. Instruction 027*ij*0 is executed in the Population/Leading Zero Count functional unit.

Instruction 027*ij*7 stores (A*i*) to the SB*j* register, which is shared between the CPUs in the same cluster.

HOLD ISSUE CONDITIONS: For instruction 027*ij*0, instruction 033 issued in previous CP

A*i* reserved

S*j* reserved (except S0)

For instruction 027*ij*7:
Shared register access conflict due to instruction 027*ij*7 or 073*ij*3 in CIP of other CPU at same CP, regardless of clustering

EXECUTION TIME: Instruction issue, 1 CP

For instruction 027*ij*0, A*i* ready, 3 CPs

For instruction 027*ij*7, SB*j* ready, 3 CPs

SPECIAL CASES: For instruction 027*ij*0, (A*i*)=64 if *j*=0.

For instruction 027*ij*0, (A*i*)=0 if (S*j*) is negative.

Instruction 027*ij*7 is a no-op if CLN=0.

CAL Syntax	Description	Octal Code
A*i* A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*	030*ijk*
A*i* A*k*†	Transmit (A*k*) to A*i*	030*i*0*k*
A*i* A*j*+1†	Integer sum of (A*j*) and 1 to A*i*	030*ij*0
A*i* A*j*-A*k*	Integer difference (A*j*) less (A*k*) to A*i*	031*ijk*
A*i* -1†	Transmit -1 to A*i*	031*i*00
A*i* -A*k*†	Transmit the negative of (A*k*) to A*i*	031*i*0*k*
A*i* A*j*-1†	Integer difference (A*j*) less 1 to A*i*	031*ij*0

Instruction 030 forms the integer sum of (A*j*) and (A*k*) and enters the result into A*i*. No overflow is detected.

Instruction 031 forms the integer difference of (A*j*) and (A*k*) and enters the result into A*i*. No overflow is detected.

Instructions 030 and 031 are executed in the Address Add functional unit.

HOLD ISSUE CONDITIONS: A*i* reserved

A*j* or A*k* reserved (except A0)

EXECUTION TIME: Instruction issue, 1 CP

A*i* ready, 2 CPs

SPECIAL CASES: For instruction 030:
(A*i*)=(A*k*) if *j*=0 and *k*≠0.
(A*i*)=1 if *j*=0 and *k*=0.
(A*i*)=(A*j*) + 1 if *j*≠0 and *k*=0.

For instruction 031:
(A*i*)= -(A*k*) if *j*=0 and *k*≠0.
(A*i*)= -1 if *j*=0 and *k*=0.
(A*i*)=(A*j*) - 1 if *j*≠0 and *k*=0.

† Special CAL syntax

INSTRUCTION 032

CAL Syntax	Description	Octal Code
A*i* A*j**A*k*	Integer product of (A*j*) and (A*k*) to A*i*	032*ijk*

Instruction 032 forms the integer product of (A*j*) and (A*k*) and enters the low-order 24 bits of the result into A*i*. No overflow is detected.

Instruction 032 is executed in the Address Multiply functional unit.

HOLD ISSUE CONDITIONS: A*i* reserved

A*j* or A*k* reserved (except A0)

EXECUTION TIME: Instruction issue, 1 CP

A*i* ready, 4 CPs

SPECIAL CASES: (A*i*)=0 if *j*=0.
(A*k*)=1 if *k*=0.
Thus, (A*i*)=(A*j*) if *j*≠0 and *k*=0.

CAL Syntax		Description	Octal Code
A*i*	CI	Channel number of highest priority interrupt request to A*i*	033*i*00
A*i*	CA,A*j*	Current address of channel (A*j*) to A*i*	033*ij*0
A*i*	CE,A*j*	Error flag of channel (A*j*) to A*i*	033*ij*1

Instruction 033 enters channel status information into A*i*. The *j* and *k* designators and the contents of A*j* define the desired information.

The channel number of the highest priority interrupt request is entered into A*i* when the *j* designator is 0. The contents of A*j* specify a channel number when the *j* designator is nonzero. The value of the Current Address (CA) register for the channel is entered into A*i* when the *k* designator is 0. The error flag for the channel is entered into the low-order bit of A*i* when the *k* designator is 1. The high-order bits of A*i* are cleared. The error flag can be cleared only in monitor mode using instruction 0012.

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

HOLD ISSUE CONDITIONS: A*i* reserved

A*j* reserved (except A0)

EXECUTION TIME: Instruction issue, 1 CP

A*i* ready, 4 CPs

SPECIAL CASES: (A*i*)=highest priority channel causing interrupt if (A*j*)=0.

(A*i*)=current address of channel (A*j*) if (A*j*)≠0 and *k*=0.

(A*i*)=I/O error flag of channel (A*j*) if (A*j*)≠0 and *k*=1.

(A*i*)=0 if (A*j*)=1.

2 CPs must elapse after instruction 0012*j*0 issues before issuing instruction 033*i*00.

SPECIAL CASES:
(continued)

When k=1:

Bits 2^{12} through 2^{20} contain the remaining block length.

Bit 2^{18} indicates a request in progress.

Bit 2^{19} indicates either an SSD single-bit memory error (during a read SSD operation) or an SSD single-bit channel error (during a write SSD operation).

Bit 2^{20} indicates a block length error.

Bit 2^{21} indicates either an SSD double-bit memory error (during a read SSD operation) or an SSD double-bit channel error (during a write SSD operation).

Bit 2^{22} indicates a CPU double-bit memory error.

Bit 2^{23} indicates a fatal error (if bit 2^{20}, 2^{21}, or 2^{22} is set).

CAL Syntax	Description	Octal Code
B*jk*,A*i* ,A0	Block transfer (A*i*) words from memory starting at address (A0) to B registers starting at register *jk*	034*ijk*
B*jk*,A*i* 0,A0 †	Block transfer (A*i*) words from memory starting at address (A0) to B registers starting at register *jk*	034*ijk*
,A0 B*jk*,A*i*	Block transfer (A*i*) words from B registers starting at register *jk* to memory starting at address (A0)	035*ijk*
0,A0 B*jk*,A*i* †	Block transfer (A*i*) words from B registers starting at register *jk* to memory starting at address (A0)	035*ijk*
T*jk*,A*i* ,A0	Block transfer (A*i*) words from memory starting at address (A0) to T registers starting at register *jk*	036*ijk*
T*jk*,A*i* 0,A0 †	Block transfer (A*i*) words from memory starting at address (A0) to T registers starting at register *jk*	036*ijk*
,A0 T*jk*,A*i*	Block transfer (A*i*) words from T registers starting at register *jk* to memory starting at address (A0)	037*ijk*
0,A0 T*jk*,A*i* †	Block transfer (A*i*) words from T registers starting at register *jk* to memory starting at address (A0)	037*ijk*

Instructions 034 through 037 perform block transfers between memory and B or T registers.

In all the instructions, the amount of data transferred is specified by the low-order 7 bits of (A*i*). 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 B77 or T77 is reached. Since processing of the registers is circular, B00 is processed after B77 and T00 is processed after T77 if the count in (A*i*) is not exhausted.

† Special CAL syntax

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 1 for successive transfers.

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

HOLD ISSUE CONDITIONS:

A0 reserved

Ai reserved

Scalar reference in CP1, CP2, or CP3

For instruction 034, Port A busy or instruction 035 in process or unidirectional memory mode and Port C busy

For instruction 035, Port C busy or instruction 034 in process or unidirectional memory mode and Port A or Port B busy

For instruction 036, Port B busy or instruction 037 in process or unidirectional memory mode and Port C busy

For instruction 037, Port C busy or instruction 036 in process or unidirectional memory mode and Port A or Port B busy

EXECUTION TIME:

Instruction issue, 1 CP

For instruction 034 or 036:
B or T register reserved 16 CPs + (Ai) if (Ai)≠0; 6 CPs if (Ai)=0.
Port A or B busy for (Ai) + 5 CPs if (Ai)≠0; 4 CPs if (Ai)=0.

For instruction 035 or 037:
B or T register reserved 5 CPs + (Ai) if (Ai)≠0; 4 CPs if (Ai)=0.
Port C busy for (Ai) + 5 CPs if (Ai)≠0; 4 CPs if (Ai)=0.

SPECIAL CASES:

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

(Ai) in the range greater than 100_8 and less than 200_8 causes a wrap-around condition.

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

NOTE

Instruction 034 uses Port A, instruction 035 uses Port C, instruction 036 uses Port B, and instruction 037 uses Port C.

CAL Syntax	Description	Octal Code
S*i* *exp*	Transmit *jkm* to S*i*	040*ijkm*
S*i* *exp*	Transmit complement of *jkm* to S*i*	041*ijkm*

The 2-parcel instructions 040 and 041 enter immediate values into an S register.

Instruction 040 enters a 64-bit value composed of the 22-bit *jkm* field and 42 high-order bits of 0 into S*i*.

Instruction 041 enters a 64-bit value that is the complement of a value formed by the 22-bit *jkm* field and 42 high-order bits of 0 into S*i*. The complement is formed by changing all 1 bits to 0 and all 0 bits to 1. Thus, for instruction 041, the high-order 42 bits of S*i* are set to 1's. The instruction provides for entering a negative value into S*i*. Since the register value is the ones complement of *jkm*, to get the twos complement *jkm* should be 0 to get -1, 1 to get -2, 3 to get -4, etc.

HOLD ISSUE CONDITIONS: S*i* reserved

Second parcel not in a buffer

EXECUTION TIME: Instruction issue:
Both parcels in same buffer, 2 CPs
Both parcels in different buffers, 4 CPs
S*i* ready, 1 CP

SPECIAL CASES: None

CAL Syntax	Description	Octal Code
S*i* <*exp*	Form *exp* bits of ones mask in S*i* from right; *jk* field gets 64-*exp*.	042*ijk*
S*i* #>*exp*†	Form *exp* bits of zeros mask in S*i* from left; *jk* field gets *exp*.	042*ijk*
S*i* 1†	Enter 1 into S*i*	042*i*77
S*i* -1†	Enter -1 into S*i*	042*i*00
S*i* >*exp*	Form *exp* bits of ones mask in S*i* from left; *jk* field gets *exp*.	043*ijk*
S*i* #<*exp*†	Form *exp* bits of zeros mask in S*i* from right; *jk* field gets 64-*exp*.	043*ijk*
S*i* 0†	Clear S*i*	043*i*00

Instruction 042 generates a mask of 64-*jk* ones from right to left in S*i*. For example, if *jk*=0, S*i* contains all 1 bits (integer value= -1) and if *jk*=77_8, S*i* contains zeros in all but the low-order bit (integer value=1).

Instruction 043 generates a mask of *jk* ones from left to right in S*i*. For example, if *jk*=0, S*i* contains all 0 bits (integer value=0) and if *jk*=77_8, S*i* contains ones in all but the low-order bit (integer value= -2).

Instructions 042 and 043 are executed in the Scalar Logical functional unit.

HOLD ISSUE CONDITIONS: S*i* reserved

EXECUTION TIME: Instruction issue, 1 CP

S*i* ready, 1 CP

SPECIAL CASES: None

† Special CAL syntax

CAL Syntax	Description	Octal Code
Si Sj&Sk	Logical product of (Sj) and (Sk) to Si	044ijk
Si Sj&SB†	Sign bit of (Sj) to Si	044ij0
Si SB&Sj†	Sign bit of (Sj) to Si (j≠0)	044ij0
Si #Sk&Sj	Logical product of (Sj) and complement of (Sk) to Si	045ijk
Si #SB&Sj†	(Sj) with sign bit cleared to Si	045ij0
Si Sj\Sk	Logical difference of (Sj) and (Sk) to Si	046ijk
Si Sj\SB†	Toggle sign bit of (Sj), then enter into Si	046ij0
Si SB\Sj†	Toggle sign bit of (Sj), then enter into Si (j≠0)	046ij0
Si #Sj\Sk	Logical equivalence of (Sk) and (Sj) to Si	047ijk
Si #Sk†	Transmit ones complement of (Sk) to Si	047i0k
Si #Sj\SB†	Logical equivalence of (Sj) and sign bit to Si	047ij0
Si #SB\Sj†	Logical equivalence of (Sj) and sign bit to Si (j≠0)	047ij0
Si #SB†	Enter ones complement of sign bit into Si	047i00
Si Sj!Si&Sk	Scalar merge	050ijk
Si Sj!Si&SB†	Scalar merge of (Si) and sign bit of (Sj) to Si	050ij0
Si Sj!Sk	Logical sum of (Sj) and (Sk) to Si	051ijk
Si Sk†	Transmit (Sk) to Si	051i0k
Si Sj!SB†	Logical sum of (Sj) and sign bit to Si	051ij0
Si SB!Sj†	Logical sum of (Sj) and sign bit to Si (j≠0)	051ij0
Si SB†	Enter sign bit into Si	051i00

† Special CAL syntax

NOTE

For instructions 044 through 051, SB with no register designator is the sign bit, not Shared Address register.

Instructions 044 through 051 are executed in the Scalar Logical functional unit.

Instruction 044 forms the logical product (AND) of (Sj) and (Sk) and enters the result into Si. Bits of Si are set to 1 when corresponding bits of (Sj) and (Sk) are 1 as in the following example:

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

(Sj) is transmitted to Si if the j and k designators have the same nonzero value. Si is cleared if the j designator is 0. The sign bit of (Sj) is transmitted to Si if the j designator is nonzero and the k designator is 0.

Instruction 045 forms the logical product (AND) of (Sj) and the complement of (Sk) and enters the result into Si. Bits of Si are set to 1 when corresponding bits of (Sj) and the complement of (Sk) are 1 as in the following example where (Sk') = complement of (Sk):

if (Sk) = 1 0 1 0

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

Si is cleared if the j and k designators have the same value or if the j designator is 0. (Sj) with the sign bit cleared is transmitted to Si if the j designator is nonzero and the k designator is 0.

Instruction 046 forms the logical difference (exclusive OR) of (Sj) and (Sk) and enters the result into Si. Bits of Si are set to 1 when 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 0 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 0.

Instruction 047 forms the logical equivalence of (Sj) and (Sk) and enters the result into Si. Bits of Si are set to 1 when 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 0 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 0. The result is the complement produced by instruction 046.

Instruction 050 merges the contents of (Sj) with (Si) depending on the ones mask in Sk. The result is defined by the following Boolean equation where Sk' is the complement of Sk as illustrated:

(Si) = (Sj)(Sk) + (Si)(Sk')

if (Sk) = 1 1 1 1 0 0 0 0

(Sk') = 0 0 0 0 1 1 1 1
(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

Instruction 050 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 1 if the j designator is 0 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 0. The sign bit of Si is cleared if the j and k designators are both 0.

Instruction 051 forms the logical sum (inclusive OR) of (Sj) and (Sk) and enters the result into Si. Bits of Si are set when 1 of the corresponding bits of (Sj) and (Sk) is set as in the following example:

(Sj) = 1 1 0 0
(Sk) = 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 0 and the k designator is nonzero. (Sj) with the sign bit set to 1 is transmitted to Si if the j designator is nonzero and the k designator is 0. A ones mask consisting of only the sign bit is entered into Si if the j and k designators are both 0.

HOLD ISSUE CONDITIONS: Si reserved

Sj or Sk reserved (except S0)

EXECUTION TIME: Instruction issue, 1 CP

Si ready, 1 CP

SPECIAL CASES: (Sj)=0 if j=0.

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

CAL Syntax	Description	Octal Code
S0 S*i*<*exp*	Shift (S*i*) left *exp*=*jk* places to S0	052*ijk*
S0 S*i*>*exp*	Shift (S*i*) right *exp*=64-*jk* places to S0	053*ijk*
S*i* S*i*<*exp*	Shift (S*i*) left *exp*=*jk* places to S*i*	054*ijk*
S*i* S*i*>*exp*	Shift (S*i*) right *exp*=64-*jk* places to S*i*	055*ijk*

Instructions 052 through 055 are executed in the Scalar Shift functional unit. They shift values in an S register by an amount specified by *jk*. All shifts are end off with zero fill.

Instruction 052 shifts (S*i*) left *jk* places and enters the result into S0. Shift range is 0 through 63 left.

Instruction 053 shifts (S*i*) right by 64-*jk* places and enters the result into S0. Shift range is 1 through 64 right.

Instruction 054 shifts (S*i*) left *jk* places and enters the result into S*i*. Shift range is 0 through 63 left.

Instruction 055 shifts (S*i*) right by 64-*jk* places and enters the result into S*i*. Shift range is 1 through 64 right.

HOLD ISSUE CONDITIONS: Instruction 056, 057, 060, or 061 issued in previous CP

S*i* reserved

For instructions 052 and 053, S0 reserved

EXECUTION TIME: Instruction issue, 1 CP

For instructions 052 and 053, S0 ready, 2 CPs

For instructions 054 and 055, S*i* ready, 2 CPs

SPECIAL CASES: None

CAL Syntax	Description	Octal Code
S*i* S*i*,S*j*<A*k*	Shift (S*i*) and (S*j*) left by (A*k*) places to S*i*	056*ijk*
S*i* S*i*,S*j*<1†	Shift (S*i*) and (S*j*) left one place to S*i*	056*ij*0
S*i* S*i*<A*k*†	Shift (S*i*) left (A*k*) places to S*i*	056*i*0*k*
S*i* S*j*,S*i*>A*k*	Shift (S*j*) and (S*i*) right by (A*k*) places to S*i*	057*ijk*
S*i* S*j*,S*i*>1†	Shift (S*j*) and (S*i*) right one place to S*i*	057*ij*0
S*i* S*i*>A*k*†	Shift (S*i*) right (A*k*) places to S*i*	057*i*0*k*

Instructions 056 and 057 are executed in the Scalar Shift functional unit. They shift 128-bit values formed by logically joining two S registers. Shift counts are obtained from register A*k*. All shift counts, (A*k*), are considered positive and all 24 bits of (A*k*) are used for the shift count. A shift of one place occurs if the *k* designator is 0. If *j*=0, the shifts function as if the shifted value was 64 bits rather than 128 bits since the S*j* value used is 0.

The shifts are circular if the shift count does not exceed 64 and the *i* and *j* designators are equal and nonzero. For instructions 056 and 057, (S*j*) is unchanged, provided *i*≠*j*. For shifts greater than 64, the shift is end off with zero fill. If *i*=*j* and the shift is greater than 64, the shift is the same as if the respective instruction 054 or 055 was used with a shift count 64 less.

Instruction 056 performs left shifts of (S*i*) and (S*j*) with (S*i*) initially the most significant bits of the double register. The high-order 64 bits of the result are transmitted to S*i*. S*i* is cleared if the shift count exceeds 127. Instruction 056 produces the same result as instruction 054 if the shift count does not exceed 63 and the *j* designator is 0.

Instruction 057 performs right shifts of (S*j*) and (S*i*) with (S*j*) initially the most significant bits of the double register. The low-order 64 bits of the result are transmitted to S*i*. S*i* is cleared if the shift count exceeds 127. Instruction 057 produces the same result as instruction 055 if the shift count does not exceed 63 and the *j* designator is 0.

† Special CAL syntax

HOLD ISSUE CONDITIONS:	S*i* reserved
	S*j* or A*k* reserved (except S0 and/or A0)
EXECUTION TIME:	Instruction issue, 1 CP
	S*i* ready, 3 CPs
SPECIAL CASES:	(S*j*)=0 if *j*=0.
	(A*k*)=1 if *k*=0.
	Circular shift if *i*=*j*≠0 and A*k* greater than or equal to 0 and less than or equal to 64.

CAL Syntax		Description	Octal Code
Si	Sj+Sk	Integer sum of (Sj) and (Sk) to Si	060ijk
Si	Sj-Sk	Integer difference of (Sj) and (Sk) to Si	061ijk
Si	-Sk†	Transmit negative of (Sk) to Si	061i0k

Instruction 060 forms the integer sums of (Sj) and (Sk) and enters the result into Si. No overflow is detected.

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

Instructions 060 and 061 are executed in the Scalar Add functional unit.

HOLD ISSUE CONDITIONS: Si reserved

Sj or Sk reserved (except S0)

EXECUTION TIME: Si ready, 3 CPs

Instruction issue, 1 CP

SPECIAL CASES: $(Si)=2^{63}$ if $j=0$ and $k=0$.

For instruction 060:
$(Si)=(Sk)$ if $j=0$ and $k\neq0$.
$(Si)=(Sj)$ with 2^{63} complemented if $j\neq0$ and $k=0$.

For instruction 061:
$(Si)=-(Sk)$ if $j=0$ and $k\neq0$.
$(Si)=(Sj)$ with 2^{63} complemented if $j\neq0$ and $k=0$.

† Special CAL syntax

CAL Syntax	Description	Octal Code
Si Sj+FSk	Floating sum of (Sj) and (Sk) to Si	062ijk
Si +FSk†	Normalize (Sk) to Si	062i0k
Si Sj-FSk	Floating difference of (Sj) and (Sk) to Si	063ijk
Si -FSk†	Transmit normalized negative of (Sk) to Si	063i0k

Instructions 062 and 063 are performed in the Floating-point Add functional unit. Operands are assumed to be in floating-point format. The result is normalized even if the operands are not normalized.

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

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

Overflow conditions are described in section 4. For floating-point operands with the sign bit set (bit=1), zero exponent and zero coefficient are treated as 0 (that is, all 64 bits=0).††

HOLD ISSUE CONDITIONS: Si reserved

Sj or Sk reserved (except S0)

Instructions 170 through 173 in process, unit busy (VL) + 4 CPs

EXECUTION TIME: Instruction issue, 1 CP

Si ready, 6 CPs

† Special CAL syntax

†† Considered -0. No floating-point unit generates a -0 except the Floating-point Multiply functional unit if one of the operands was a -0. Normally, -0 occurs in logical manipulations when a sign is attached to a number; that number can be 0.

SPECIAL CASES:

For instruction 062:
(S*i*)=(S*k*) normalized if (S*k*) exponent is valid, *j*=0 and *k*≠0.
(S*i*)=(S*j*) normalized if (S*j*) exponent is valid, *j*≠0 and *k*=0.

For instruction 063:
(S*i*)= -(S*k*) normalized if (S*k*) exponent is valid, *j*=0 and *k*≠0. Sign of (S*i*) is opposite that of (S*k*) if (S*k*)≠0.
(S*i*)=(S*j*) normalized if (S*j*) exponent is valid, *j*≠0 and *k*=0.

CAL Syntax		Description	Octal Code
S*i*	S*j**FS*k*	Floating-point product of (S*j*) and (S*k*) to S*i*	064*ijk*
S*i*	S*j**HS*k*	Half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*	065*ijk*
S*i*	S*j**RS*k*	Rounded floating-point product of (S*j*) and (S*k*) to S*i*	066*ijk*
S*i*	S*j**IS*k*	Reciprocal iteration; 2-(S*j*)*(S*k*) to S*i*	067*ijk*

Instructions 064 through 067 are executed in the Floating-point Multiply functional unit. Operands are assumed to be in floating-point format. The result is not guaranteed to be normalized if the operands are not normalized.

Instruction 064 forms the product of the floating-point quantities in S*j* and S*k* and enters the result into S*i*.

Instruction 065 forms the half-precision rounded product of the floating-point quantities in S*j* and S*k* and enters the result into S*i*. The low-order 19 bits of the result are cleared.

Instruction 066 forms the rounded product of the floating-point quantities in S*j* and S*k* and enters the result into S*i*.

Instruction 067 forms two minus the product of the floating-point quantities in S*j* and S*k* and enters the result into S*i*. This instruction is used in the divide sequence as described in section 4 under Floating-point Arithmetic.

In the evaluation C = 2-B*A, B must be a reciprocal of A of less than 47 significant bits and not the exact reciprocal, otherwise C will be in error. The reciprocal produced by the reciprocal approximation instruction meets this criterion.

HOLD ISSUE CONDITIONS: S*i* reserved

S*j* or S*k* reserved (except S0)

Instructions 160 through 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.

If both exponent fields are 0, an integer multiply is performed. Correct integer multiply results are produced if the following conditions are met:

- Both operand sign bits are 0.
- The sum of the 0 bits to the right of the least significant 1 bit in the two operands is greater than or equal to 48.

The integer result obtained is the high-order 48 bits of the 96-bit product of the two operands.

CAL Syntax	Description	Octal Code
S*i* /HS*j*	Floating-point reciprocal approximation of (S*j*) to S*i*	070*ij*0

Instruction 070 is executed in the Reciprocal Approximation functional unit.

Instruction 070 forms an approximation to the reciprocal of the normalized floating-point quantity in S*j* and enters the result into S*i*. This instruction occurs in the divide sequence to compute the quotient of two floating-point quantities as described in section 4 under Floating-point Arithmetic.

The reciprocal approximation instruction produces a result of 30 significant bits. The low-order 18 bits are zeros. The number of significant bits can be extended to 48 using the reciprocal iteration instruction and a multiply.

HOLD ISSUE CONDITIONS:

S*i* reserved

S*j* reserved (except S0)

Instruction 174 in process, unit busy (VL) + 4 CPs

EXECUTION TIME:

S*i* ready, 14 CPs

Instruction issue, 1 CP

SPECIAL CASES:

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

(S*j*)=0 produces a range error; the result is meaningless.

(S*j*)=0 if *j*=0.

CAL Syntax		Description	Octal Code
S*i*	A*k*	Transmit (A*k*) to S*i* with no sign extension	071*i*0*k*
S*i*	+A*k*	Transmit (A*k*) to S*i* with sign extension	071*i*1*k*
S*i*	+FA*k*	Transmit (A*k*) to S*i* as unnormalized floating-point number	071*i*2*k*
S*i*	0.6	Transmit constant 0.75 x 2^{48} to S*i*	071*i*30
S*i*	0.4	Transmit constant 0.5 to S*i*	071*i*40
S*i*	1.	Transmit constant 1.0 to S*i*	071*i*50
S*i*	2.	Transmit constant 2.0 to S*i*	071*i*60
S*i*	4.	Transmit constant 4.0 to S*i*	071*i*70

Instruction 071 performs functions that depend on the value of the *j* designator. The functions are concerned with transmitting information from an A register to an S register and with generating frequently used floating-point constants.

When the *j* designator is 0, the 24-bit value in A*k* is transmitted to S*i*. The value is treated as an unsigned integer. The high-order bits of S*i* are zeros.

When the *j* designator is 1, the 24-bit value in A*k* is transmitted to S*i*. The value is treated as a signed integer. The sign bit of A*k* is extended through the high-order bit of S*i*.

When the *j* designator is 2, the 24-bit value in A*k* is transmitted to S*i* as an unnormalized floating-point quantity (the result is then added to 0 to normalize). For this instruction, the exponent in bits 2^{62} through 2^{48} is set to 40060_8. The sign of the coefficient is set according to the sign of A*k*. If the sign bit of A*k* is set, the twos complement of A*k* is entered into S*i* as the magnitude of the coefficient and bit 2^{63} of S*i* is set for the sign of the coefficient.

A sequence of instructions is used to convert to floating-point format an integer whose absolute value is less than 24 bits:

```
CAL code:   A1   S1
            S1   +FA1
            S1   +FS1     9 CPs required
```

When the j designator is 3, the floating-point constant of 0.75×2^{48} is entered into Si ($0\ 40060\ 6000\ 0000\ 0000\ 0000_8$). This constant is used to create floating-point numbers from integer numbers (positive and negative) whose absolute value is less than 47 bits. A sequence of instructions is used for conversion of an integer in S1:

```
CAL code:   S2  0.6
            S1  S2-S1
            S1  S2-FS1      11 CPs required
```

When the j designator is 4, the floating-point constant 0.5 (= $0\ 40000\ 4000\ 0000\ 0000\ 0000_8$) is entered into S$i$.

When the j designator is 5, the floating-point constant 1.0 (= $0\ 40001\ 4000\ 0000\ 0000\ 0000_8$) is entered into S$i$.

When the j designator is 6, the floating-point constant 2.0 (= $0\ 40002\ 4000\ 0000\ 0000\ 0000_8$) is entered into S$i$.

When the j designator is 7, the floating-point constant 4.0 (= $0\ 40003\ 4000\ 0000\ 0000\ 0000_8$) is entered into S$i$.

HOLD ISSUE CONDITIONS: Si reserved

Ak reserved (except A0); applies to all forms of the instruction, that is, j designators 0 through 7.

EXECUTION TIME: Instruction issue, 1 CP

Si ready, 2 CPs

SPECIAL CASES: (Ak)=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 x 2^{60} (octal) if j=3.

(Si)=0.4 x 2^{0} (octal) if j=4.

(Si)=0.4 x 2^{1} (octal) if j=5.

(Si)=0.4 x 2^{2} (octal) if j=6.

(Si)=0.4 x 2^{3} (octal) if j=7.

CAL Syntax	Description	Octal Code
S*i* RT	Transmit (RTC) to S*i*	072*i*00
S*i* SM	Read semaphores to S*i*	072*i*02
S*i* ST*j*	Read (ST*j*) register to S*i*	072*ij*3
S*i* VM	Transmit (VM) to S*i*	073*i*00
S*i* SR*j*	Transmit (SR*j*) to S*i*; *j*=0	073*ij*1
SM S*i*	Load semaphores from S*i*	073*i*02
ST*j* S*i*	Load (ST*j*) register from S*i*	073*ij*3
S*i* T*jk*	Transmit (T*jk*) to S*i*	074*ijk*
T*jk* S*i*	Transmit (S*i*) to T*jk*	075*ijk*

Instruction 072*i*00 enters the 64-bit value of the real-time clock (RTC) into S*i*. The clock is incremented by 1 each CP. The RTC can be set only by the monitor through use of instruction 0014*j*0.

Instruction 072*i*02 enters the values of all of the semaphores into S*i*. The 32-bit SM register is left justified in S*i* with SM00 occupying the sign bit.

Instruction 072*ij*3 enters the contents of ST*j* into S*i*.

Instruction 073*i*00 enters the 64-bit value of the VM register into S*i*. The VM register is usually read after being set by instruction 175.

Instruction 073*ij*1 enters the contents of the Status register SR*j* into S*i*. Instruction 073*i*01 returns the following status to the high-order bits of S*i*:

Status	Bit	Description
2	2^{62}	Processor number (PN)
1	2^{60}	Program state (PS)
3	2^{57}, 2^{58}	Cluster number (CN)
1	2^{56}	Floating-point interrupts enabled (IFP)
1	2^{55}	Floating-point error occurred (FPS)
1	2^{54}	Bidirectional memory enabled (BDM)
1	2^{53}	Operand range interrupts enabled (IOR)

Instruction 073*i*02 sets the semaphores from 32 high-order bits of S*i*. SM00 receives the sign bit of S*i*.

Instruction 073*ij*3 enters the contents of S*i* into ST*j*.

Instruction 074 enters the contents of T*jk* into S*i*.

Instruction 075 enters the contents of S*i* into T*jk*.

HOLD ISSUE CONDITIONS: S*i* reserved

For instructions 074 and 075, instructions 036 through 037 in process

For instruction 074, instruction 075 issued in the previous CP

For instruction 073*i*00:
Instruction 14*x* or 175 in process, VM busy for (VL) + 5 CPs
Instruction 003 in process, VM busy for 1 CP

For instructions 072*ij*3 and 073*ij*3 shared register access conflict due to:
Instruction 072*ij*3 or 026*ij*7 in CIP of other CPU, regardless of clustering, at the same CP; or
Instruction 073*ij*3 or 027*ij*7 issued by both CPUs, regardless of clustering, at 3 CPs earlier.

For instruction 073*i*02, hold issue for 4 CPs unconditionally.

EXECUTION TIME: Instruction issue, 1 CP

All cases except 073*ij*3, result register ready, 1 CP

For 073*ij*3, ST*j* ready, 3 CPs

For 073*i*02, SM ready, 1 CP

SPECIAL CASES: For instructions 072*i*02 and 072*ij*3, (S*i*)=0 if CLN=0.

Instructions 073*i*02 and 073*ij*3 are no-ops if CLN=0.

CAL Syntax	Description	Octal Code
S*i* V*j*,A*k*	Transmit (V*j* element (A*k*)) to S*i*	076*ijk*
V*i*,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*)	077*ijk*
V*i*,A*k* 0†	Clear V*i* element (A*k*)	077*i*0*k*

Instructions 076 and 077 transmit a 64-bit quantity between a V register element and an S register.

Instruction 076 transmits the contents of an element of register V*j* to S*i*.

Instruction 077 transmits the contents of register S*j* to an element of register V*i*.

The low-order 6 bits of (A*k*) determine the vector element for either instruction.

HOLD ISSUE CONDITIONS: A*k* reserved (except A0)

For instruction 076, S*i* reserved or V*j* reserved as operand or as result

For instruction 077, V*i* reserved as operand or as result or S*j* reserved

EXECUTION TIME: Instruction issue, 1 CP

For instruction 076, S*i* ready, 4 CPs

For instruction 077, V*i* ready, 1 CP

SPECIAL CASES: (S*j*)=0 if *j*=0.

(A*k*)=1 if *k*=0.

† Special CAL syntax

CAL Syntax	Description	Octal Code
A*i* *exp*,A*h*	Read from ((A*h*) + *jkm*) to A*i*	10*hijkm*
A*i* *exp*,0†	Read from (*jkm*) to A*i*	100*ijkm*
A*i* *exp*,†	Read from (*jkm*) to A*i*	100*ijkm*
A*i* ,A*h*†	Read from (A*h*) to A*i*	10*hi*00 0
exp,A*h* A*i*	Store (A*i*) to (A*h*) + *jkm*	11*hijkm*
exp,0 A*i*†	Store (A*i*) to *jkm*	110*ijkm*
exp, A*i*†	Store (A*i*) to *exp*	110*ijkm*
,A*h* A*i*†	Store (A*i*) to (A*h*)	11*hi*00 0
S*i* *exp*,A*h*	Read from ((A*h*) + *jkm*) to S*i*	12*hijkm*
S*i* *exp*,0†	Read from (*exp*) to S*i*	120*ijkm*
S*i* *exp*,†	Read from (*exp*) to S*i*	120*ijkm*
S*i* ,A*h*†	Read from (A*h*) to S*i*	12*hi*00 0
exp,A*h* S*i*	Store (S*i*) to (A*h*) + *jkm*	13*hijkm*
exp,0 S*i*†	Store (S*i*) to *exp*	130*ijkm*
exp, S*i*†	Store (S*i*) to *exp*	130*ijkm*
,A*h* S*i*†	Store (S*i*) to (A*h*)	13*hi*00 0

The 2-parcel instructions 10*h* through 13*h* transmit data between memory and an A register or an S register. The content of A*h* (treated as a 22-bit signed integer) is added to the signed 22-bit integer in the *jkm* field to determine the memory address. If *h* is 0, (A*h*) 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 functional unit.

† Special CAL syntax

Instructions 10*h* and 11*h* transmit 24-bit quantities to or from A registers. When transmitting data from memory to an A register, the high-order 40 bits of the memory word are ignored. On a store from A*i* into memory, the high-order 40 bits of the memory word are zeroed.

Instructions 12*h* and 13*h* transmit 64-bit quantities to or from register S*i*.

HOLD ISSUE CONDITIONS:

Port A, B, or C busy

A*h* reserved or busy previous CP

For instructions 10*h* and 11*h*, A*i* reserved

For instructions 12*h* and 13*h*, S*i* reserved

Instructions 10*x* through 13*x* in CP 2 and CP 3 and conflict

Second parcel not in a buffer

Second parcel in different buffer, 2 CP

EXECUTION TIME:

Instruction issue:
Both parcels in same buffer, 2 CPs
For instruction 10*h*, A*i* ready, 14 CPs
For instruction 12*h*, S*i* ready, 14 CPs
Bank ready for next scalar read or store, 4 CPs

NOTE

After issuing instructions 10*h* through 13*h*, attempting to issue instructions 034 through 037, 176, or 177 causes Ports A, B, or C to be considered busy for 4 CPs (plus additional CPs if there are conflicts).

SPECIAL CASES:

None

CAL Syntax	Description	Octal Code
Vi Sj&Vk	Logical products of (Sj) and (Vk elements) to Vi elements	140ijk
Vi Vj&Vk	Logical products of (Vj elements) and (Vk elements) to Vi elements	141ijk
Vi Sj!Vk	Logical sums of (Sj) and (Vk elements) to Vi elements	142ijk
Vi Vk†	Transmit (Vk elements) to Vi elements	142i0k
Vi Vj!Vk	Logical sums of (Vj elements) and (Vk elements) to Vi elements	143ijk
Vi Sj Vk	Logical differences of (Sj) and (Vk elements) to Vi elements	144ijk
Vi Vj Vk	Logical differences of (Vj elements) and (Vk elements) to Vi elements	145ijk
Vi 0†	Clear Vi elements	145iii
Vi Sj!Vk&VM	If VM bit=1, transmit (Sj) to the corresponding element in Vi If VM bit=0, transmit the (corresponding Vk element) to the (corresponding Vi element)	146ijk
Vi #VM&Vk†	If VM bit=1, transmit (0) to the corresponding element in Vi If VM bit=0, transmit the (corresponding Vk element) to the (corresponding Vi element)	146i0k
Vi Vj!Vk&VM	If VM bit=1, transmit the (corresponding Vj element) to the (corresponding Vi element) If VM bit=0, transmit the (corresponding Vk element) to the (corresponding Vi element)	147ijk

Instructions 140 through 147 are executed in the Vector Logical functional unit. The number of operations performed is determined by the contents of the VL register. All operations start with element 0 of the

† Special CAL syntax

V*i*, V*j*, or V*k* register and increment the element number by 1 for each operation performed. All results are delivered to V*i*.

For instructions 140, 142, 144, and 146, a copy of the content of S*j* is delivered to the functional unit. The copy of the content is held as one of the operands until completion of the operation. Therefore, S*j* can be changed immediately without affecting the vector operation. For instructions 141, 143, 145, and 147, all operands are obtained from V registers.

Instructions 140 and 141 form the logical products (AND) of operand pairs and enter the result into V*i*. Bits of an element of V*i* are set to 1 when the corresponding bits of (S*j*) or (V*j* element) and (V*k* element) are 1 as in the following:

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

Instructions 142 and 143 form the logical sums (inclusive OR) of operand pairs and deliver the results to V*i*. Bits of an element of V*i* are set to 1 when one of the corresponding bits of (S*j*) or (V*j* element) and (V*k* element) is 1 as in the following:

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

Instructions 144 and 145 form the logical differences (exclusive OR) of operand pairs and deliver the results of V*i*. Bits of an element are set to 1 when the corresponding bit of (S*j*) or (V*j* element) is different from (V*k* 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
```

Instructions 146 and 147 transmit operands to V*i* depending on the contents of the VM register. Bit 2^{63} of the mask corresponds to element 0 of a V register. Bit 2^{0} corresponds to element 63. Operand pairs used for the selection depend on the instruction. For instruction 146, the first operand is always (S*j*), the second operand is (V*k* element). For instruction 147, the first operand is (V*j* element) and the second operand is (V*k* element). If bit *n* of the vector mask is 1, the first operand is transmitted; if bit *n* of the mask is 0, the second operand, (V*k* element), is selected.

Examples:

1. If instruction 146 is to be executed and the following register conditions exist:

```
(VL)   = 4
(VM)   = 0 60000 0000 0000 0000 0000
(S2)   = -1
(V600) = 1
(V601) = 2
(V602) = 3
(V603) = 4
```

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

```
(V700) = 1
(V701) = -1
(V702) = -1
(V703) = 4
```

The remaining elements of V7 are unaltered.

2. If instruction 147 is to be executed and the following register conditions exist:

```
(VL)   = 4
(VM)   = 0 600000 0000 0000 0000 0000
(V200) = 1      (V300) = -1
(V201) = 2      (V301) = -2
(V202) = 3      (V302) = -3
(V203) = 4      (V303) = -4
```

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

```
(V100) = -1
(V101) = 2
(V102) = 3
(V103) = -4
```

The remaining elements of V1 are unaltered.

HOLD ISSUE CONDITIONS:	Vk reserved as operand
	Vi reserved as operand or result
	Instruction 14x in process, unit busy (VL) + 4 CPs†
	Instruction 175 in process, unit busy (VL) + 4 CPs†
	For instructions 140, 142, 144, and 146, Sj reserved
	For instructions 141, 143, 145, and 147, Vj reserved as operand
EXECUTION TIME:	Instruction issue, 1 CP
	Vj or Vk ready in (VL) + 3 CPs if data available†
	Vi ready in (VL) + 7 CPs if data available†
	Unit ready, (VL) + 4 CPs if data available†
SPECIAL CASES:	$(Sj)=0$ if $j=0$.

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

CAL Syntax		Description	Octal Code
V*i*	V*j*<A*k*	Shift (V*j*) elements left by (A*k*) places to V*i* elements	150*ijk*
V*i*	V*j*<1†	Shift (V*j*) elements left one place to V*i* elements	150*ij*0
V*i*	V*j*>A*k*	Shift (V*j*) elements right by (A*k*) places to V*i* elements	151*ijk*
V*i*	V*j*>1†	Shift (V*j*) elements right one place to V*i* elements	151*ij*0

Instructions 150 and 151 are executed in the Vector Shift functional unit. The number of operations performed is determined by the contents of the VL register. Operations start with element 0 of the V*i* and V*j* registers and end with elements specified by (VL)-1.

All shifts are end off with zero fill. The shift count is obtained from (A*k*) and all 24 bits of A*k* are used for the shift count. Elements of V*i* are cleared if the shift count exceeds 63. All shift counts (A*k*) are considered positive.

Unlike shift instructions 052 through 055, these instructions receive the shift count from A*k*, rather than the *jk* fields.

HOLD ISSUE CONDITIONS: V*j* reserved as operand

V*i* reserved as operand or result

A*k* reserved (except A0)

Instructions 150 through 153 in process, unit busy (VL) + 4 CPs††

† Special CAL syntax

†† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

EXECUTION TIME: Vj ready in (VL) + 3 CPs if data available†

Vi ready in (VL) + 8 CPs if data available†

Unit ready, (VL) + 4 CPs if data available†

SPECIAL CASES: (Ak)=1 if k=0.

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

CAL Syntax	Description	Octal Code
V*i* V*j*,V*j*<A*k*	Double shifts of (V*j* elements) left (A*k*) places to V*i* elements	152*ijk*
V*i* V*j*,V*j*<1†	Double shifts of (V*j* elements) left one place to V*i* elements	152*ij*0
V*i* V*j*,V*j*>A*k*	Double shifts of (V*j* elements) right (A*k*) places to V*i* elements	153*ijk*
V*i* V*j*,V*j*>1†	Double shifts of (V*j* elements) right one place to V*i* elements	153*ij*0

Instructions 152 and 153 are executed in the Vector Shift functional unit. The instructions shift 128-bit values formed by logically joining the contents of two elements of the V*j* register. The direction of the shift determines whether the high-order bits or the low-order bits of the result are sent to V*i*. Shift counts are obtained from register A*k*.

All shifts are end off with zero fill.

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

Instruction 152 performs left shifts. The operation starts with element 0 of V*j*. If (VL) is 1, element 0 is joined with 64 bits of 0, and the resulting 128-bit quantity is then shifted left by the amount specified by (A*k*). Only the one operation is performed. The 64 high-order bits remaining are transmitted to element 0 of V*i*.

If (VL) is 2, the operation starts with element 0 of V*j* being joined with element 1, and the resulting 128-bit quantity is then shifted left by the amount specified by (A*k*). The high-order 64 bits remaining are transmitted to element 0 of V*i*. Figure 5-6 illustrates this operation.

If (VL) is greater than 2, the operation continues by joining element 1 with element 2 and transmitting the 64-bit result to element 1 of V*i*. Figure 5-7 illustrates this operation.

If (VL) is 2, element 1 is joined with 64 bits of 0 and only two operations are performed. In general, the last element of V*j* as determined by (VL) is joined with 64 bits of zeros. Figure 5-8 illustrates this operation.

† Special CAL syntax

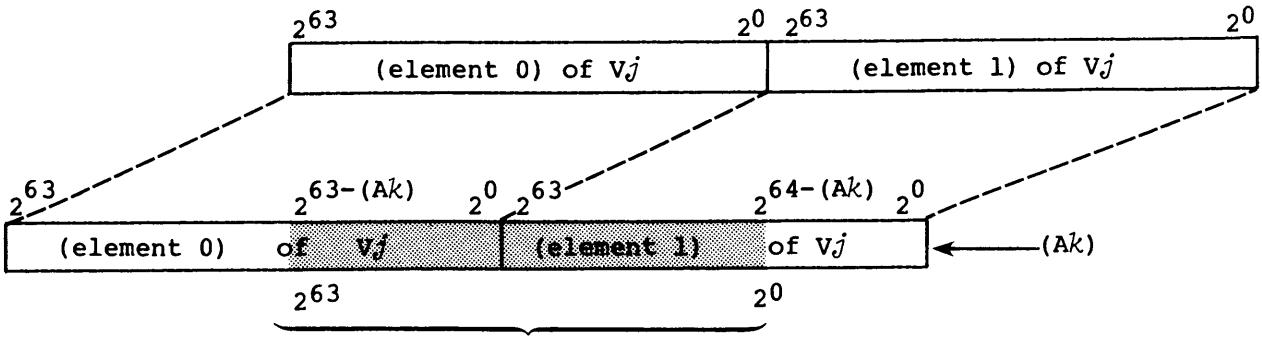


Figure 5-6. Vector left double shift, first element, VL greater than 1

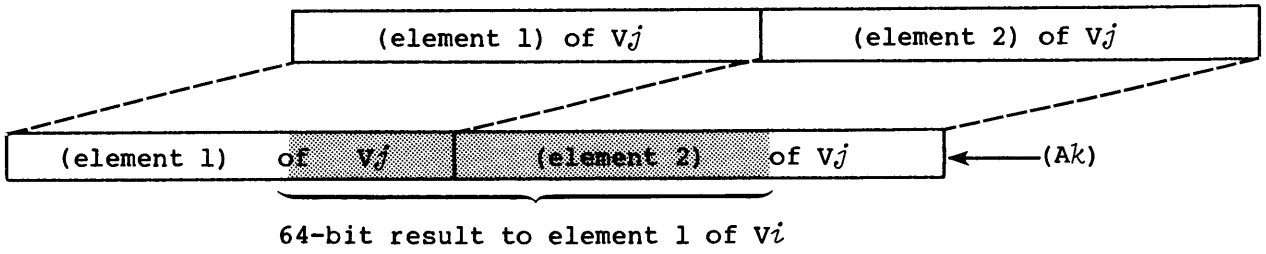


Figure 5-7. Vector left double shift, second element, VL greater than 2

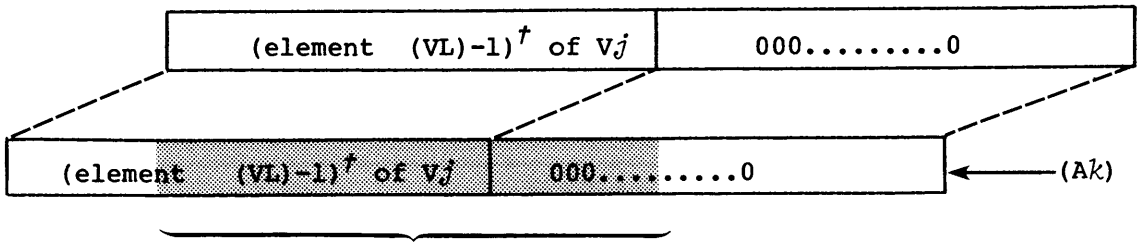


Figure 5-8. Vector left double shift, last element

† Elements are numbered 0 through 63 in the V registers; therefore, element (VL)-1 refers to the VLth element.

If (A*k*) is greater than or equal to 128, the result is all zeros. If (A*k*) is greater than 64, the result register contains at least (A*k*) - 64 zeros.

Examples:

1. If instruction 152 is to be executed and the following register conditions exist:

   (VL) = 4
   (A1) = 3
   (V400) = 0 00000 0000 0000 0000 0007
   (V401) = 0 60000 0000 0000 0000 0005
   (V402) = 1 00000 0000 0000 0000 0006
   (V403) = 1 60000 0000 0000 0000 0007

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

   (V500) = 0 00000 0000 0000 0000 0073
   (V501) = 0 00000 0000 0000 0000 0054
   (V502) = 0 00000 0000 0000 0000 0067
   (V503) = 0 00000 0000 0000 0000 0070

   Instruction 153 performs right shifts. The original element 0 of V*j* is joined with 64 high-order bits of 0 and the 128-bit quantity is shifted right by the amount specified by (A*k*). The 64 low-order bits of the result are transmitted to element 0 of V*i*. Figure 5-9 illustrates this operation.

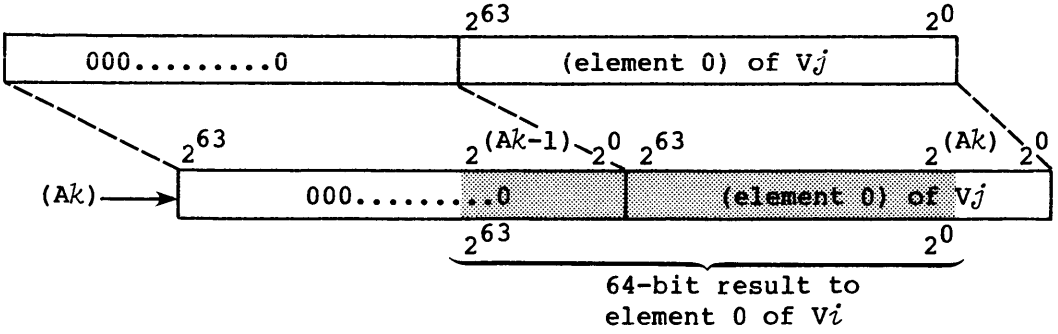


Figure 5-9. Vector right double shift, first element

If (VL)=1, only one operation is performed. In general, 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 in figure 5-10.

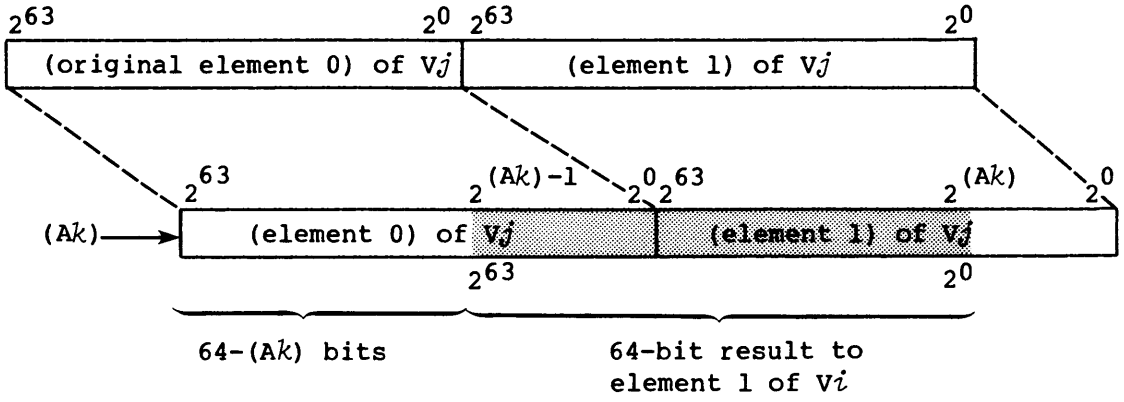


Figure 5-10. Vector right double shift, second element, VL greater than 1

The last operation performed by the instruction joins the last element of Vj as determined by (VL) with the preceding element. Figure 5-11 illustrates this operation.

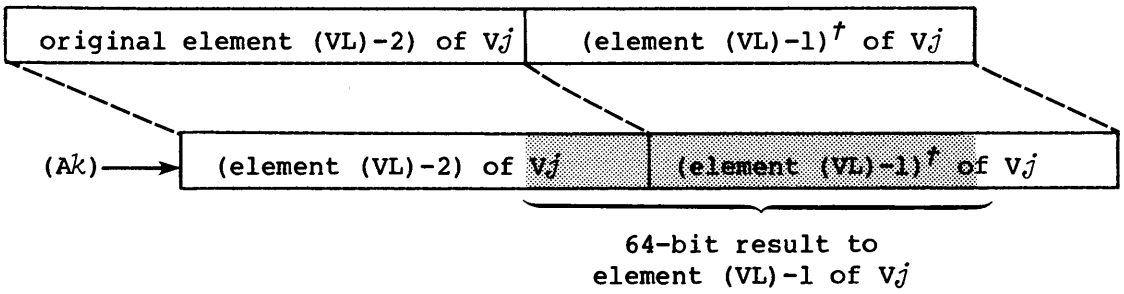


Figure 5-11. Vector right double shift, last operation

2. If an instruction 153 is to be executed and the following register conditions exist:

† Elements are numbered 0 through 63 in the V registers; therefore, element (VL)-1 refers to the VLth element.

```
(VL)   = 4
(A6)   = 3
(V200) = 0 00000 0000 0000 0000 0017
(V201) = 0 60000 0000 0000 0000 0006
(V202) = 1 00000 0000 0000 0000 0006
(V203) = 1 60000 0000 0000 0000 0007
```

Instruction 153026 is executed and following execution, register V0 contains the following values:

```
(V000) = 0 00000 0000 0000 0000 0001
(V001) = 1 66000 0000 0000 0000 0000
(V002) = 1 50000 0000 0000 0000 0000
(V003) = 1 56000 0000 0000 0000 0000
```

The remaining elements of V0 are unaltered.

HOLD ISSUE CONDITIONS:

- V*j* reserved as operand
- V*i* reserved as operand or result
- A*k* reserved (except A0)
- Instructions 150 through 153 in process, unit busy (VL) + 4 CPs†

EXECUTION TIME:

- Instruction issue, 1 CP
- V*j* ready in (VL) + 3 CPs if data available†
- For instruction 152, V*i* ready in (VL) + 9 CPs if data available†
- Instruction 153, V*i* ready in (VL) + 8 CPs if data available†
- Unit ready, (VL) + 4 CPs if data available†

SPECIAL CASES:

- (A*k*)=1 if *k*=0.

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

CAL Syntax	Description	Octal Code
V*i* S*j*+V*k*	Integer sums of (S*j*) and (V*k* elements) to V*i* elements	154*ijk*
V*i* V*j*+V*k*	Integer sums of (V*j* elements) and (V*k* elements) to V*i* elements	155*ijk*
V*i* S*j*-V*k*	Integer differences of (S*j*) and (V*k* elements) to V*i* elements	156*ijk*
V*i* -V*k*†	Transmit negative of (V*k* elements) to V*i* elements	156*i*0*k*
V*i* V*j*-V*k*	Integer differences of (V*j* elements) and (V*k* elements) to V*i* elements	157*ijk*

Instructions 154 through 157 are executed in the Vector Add functional unit.

Instructions 154 and 155 perform integer addition. Instructions 156 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 0 of the V registers and increment the element number by 1 for each operation performed. All results are delivered to elements of V*i*. No overflow is detected.

Instructions 154 and 156 deliver a copy of (S*j*) to the functional unit where the copy is retained as one of the operands until the vector operation completes. The other operand is an element of V*k*. For instructions 155 and 157, both operands are obtained from V registers.

HOLD ISSUE CONDITIONS:

V*k* reserved as operand

V*i* reserved as operand or result

Instructions 154 through 157 in process, unit busy (VL) + 4 CPs†

For instructions 154 and 156, S*j* reserved (except S0)

For instructions 155 and 157, V*j* reserved as operand

† Special CAL syntax

EXECUTION TIME:

Instruction issue, 1 CP

Vj or Vk ready in (VL) + 3 CPs if data available†

Vi ready in (VL) + 8 CPs if data available†

Unit ready, (VL) + 4 CPs if data available†

SPECIAL CASES:

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

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

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

CAL Syntax	Description	Octal Code
V*i* S*j**FV*k*	Floating-point products of (S*j*) and (V*k* elements) to V*i* elements	160*ijk*
V*i* V*j**FV*k*	Floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements	161*ijk*
V*i* S*j**HV*k*	Half-precision rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements	162*ijk*
V*i* V*j**HV*k*	Half-precision rounded floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements	163*ijk*
V*i* S*j**RV*k*	Rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements	164*ijk*
V*i* V*j**RV*k*	Rounded floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements	165*ijk*
V*i* S*j**IV*k*	Reciprocal iterations; 2 - (S*j*) * (V*k* elements) to V*i* elements	166*ijk*
V*i* V*j**IV*k*	Reciprocal iterations; 2 - (V*j* elements) * (V*k* elements) to V*i* elements	167*ijk*

Instructions 160 through 167 are executed in the Floating-point Multiply functional unit. The number of operations performed by an instruction is determined by the contents of the VL register. All operations start with element 0 of the V registers and increment the element number by 1 for each successive operation.

Operands are assumed to be in floating-point format. Instructions 160, 162, 164, and 166 deliver a copy of (S*j*) to the functional unit where the copy is retained as one of the operands until the completion of the operation. Therefore, S*j* can be changed immediately without affecting the vector operation. The other operand is an element of V*k*. For instructions 161, 163, 165, and 167, both operands are obtained from V registers.

All results are delivered to elements of V*i*. If either operand is not normalized, there is no guarantee that the products will be normalized. If neither operand is normalized, the product will not be normalized.

Out-of-range conditions are described in section 4.

Instruction 160 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.

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

Instruction 162 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 19 bits of the result elements are zeroed.

Instruction 163 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 19 bits of the result elements are zeroed.

Instruction 164 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.

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

Instruction 166 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. See the description of instruction 067 for more details.

Instruction 167 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. See the description of instruction 067 for more details.

HOLD ISSUE CONDITIONS: Vk reserved as operand

Vi reserved as operand or result

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

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

HOLD CONDITIONS: (continued)	For instructions 160, 162, 164, and 166, Sj reserved (except S0)
	For instructions 161, 163, 165, and 167, Vj reserved as operand
EXECUTION TIME:	Instruction issue, 1 CP
	Vj and Vk ready in (VL) + 3 CPs if data available†
	Vi ready in (VL) + 12 CPs if data available†
	Unit ready, (VL) + 4 CPs if data available†
SPECIAL CASES:	(Sj)=0 if j=0.

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

CAL Syntax		Description	Octal Code
V*i*	S*j*+FV*k*	Floating-point sums of (S*j*) and (V*k* elements) to V*i* element	170*ijk*
V*i*	+FV*k*†	Transmit normalized (V*k* elements) to V*i* elements	170*i*0*k*
V*i*	V*j*+FV*k*	Floating-point sums of (V*j* elements) and (V*k* elements) to V*i* elements	171*ijk*
V*i*	S*j*-FV*k*	Floating-point differences of (S*j*) and (V*k* elements) to V*i* elements	172*ijk*
V*i*	-FV*k*†	Transmit normalized negatives of (V*k* elements) to V*i* elements	172*i*0*k*
V*i*	V*j*-FV*k*	Floating-point differences of (V*j* elements) and (V*k* elements) to V*i* elements	173*ijk*

Instructions 170 through 173 are executed in the Floating-point Add functional 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 contents of the VL register. All operations start with element 0 of the V registers and increment the element number by 1 for each operation performed. All results are delivered to V*i* normalized and results are normalized even if the operands are not normalized.

Instructions 170 and 172 deliver a copy of (S*j*) to the functional unit where it remains as one of the operands until the completion of the operation. The other operand is an element of V*k*. For instructions 171 and 173, both operands are obtained from V registers. Out-of-range conditions are described in section 4.

HOLD ISSUE CONDITIONS: V*k* reserved as operand

V*i* reserved as operand or result

† Special CAL syntax

HOLD ISSUE CONDITIONS: (continued)	Instructions 170 through 173 in process, unit busy (VL) + 4 CPs†
	For instructions 170 and 172, S*j* reserved (except S0)
	For instructions 171 and 173, V*j* reserved as operand
EXECUTION TIME:	Instruction issue, 1 CP
	V*j* and V*k* ready in (VL) + 3 CPs if data available†
	V*i* ready in (VL) + 11 CPs if data available†
	Unit ready, (VL) + 4 CPs if data available†
SPECIAL CASES:	(S*j*)=0 if *j*=0.

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

CAL Syntax	Description	Octal Code
Vi /HVj	Floating-point reciprocal approximation of (Vj elements) to Vi elements	174ij0

Instruction 174 is executed in the Reciprocal Approximation functional 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.

Instruction 174 occurs in the divide sequence to compute the quotients of floating-point quantities as described in section 4 under floating-point arithmetic.

The reciprocal approximation instruction produces results of 30 significant bits. The low-order 18 bits are zeros. The number of significant bits can be extended to 48 using the reciprocal iteration instruction and a multiply.

HOLD ISSUE CONDITIONS:

Vi reserved as operand or result

Vj reserved as operand

Instruction 174 in process, unit busy for (VL) + 4 CPs†

EXECUTION TIME:

Instruction issue, 1 CP

Vj ready in (VL) + 3 CPs if data available†

Vi ready in (VL) + 19 CPs if data available†

Unit ready, (VL) + 4 CPs if data available†

SPECIAL CASES:

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

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

INSTRUCTIONS 174*ij*1 - 174*ij*2

CAL Syntax	Description	Octal Code
V*i* PV*j*	Population count of (V*j* elements) to V*i* elements	174*ij*1
V*i* QV*j*	Population count parity of (V*j* elements) to V*i* elements	174*ij*2

Instructions 174*ij*1 and 174*ij*2 are executed in the Vector Population/Parity functional unit, sharing some logic with the Reciprocal Approximation functional unit.

Instruction 174*ij*1 counts the number of bits set to 1 in each element of V*j* and enters the results into corresponding elements of V*i*. The results are entered into the low-order 7 bits of each V*i* element; the remaining high-order bits of each V*i* element are zeroed.

Instruction 174*ij*2 counts the number of bits set to 1 in each element of V*j*. 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 V*i*. The remainder of the element is set to zeros. The actual population count results are not transferred.

HOLD ISSUE CONDITIONS:

V*i* reserved as operand or result

V*j* reserved as operand

Instructions 174*xx*1 and 174*xx*2 in process, unit busy for (VL) + 4 CPs†

Instruction 174*xx*0 in process, unit busy for (VL) + 9 CPs†

Instruction 070 in process, unit busy (070 issue time) + 7 CPs†

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

EXECUTION TIME:

Instruction issue, 1 CP

V*j* ready in (VL) + 3 CPs if data available†

V*i* ready in (VL) + 10 CPs if data available†

Unit ready, (VL) + 4 CPs if data available†

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

CAL Syntax	Description	Octal Code
VM Vj,Z	VM=1 when (Vj element)=0	1750j0
VM Vj,N	VM=1 when (Vj element)≠0	1750j1
VM Vj,P	VM=1 when (Vj element) positive, (bit 2^{63}=0), includes (Vj element)=0	1750j2
VM Vj,M	VM=1 when (Vj element) negative, (bit 2^{63}=1)	1750j3

Vector mask instruction 175 is executed in the Vector Logical functional unit.

Instruction 1750*jk* creates a vector mask in VM based on the results of testing the contents of the elements of register V*j*. Each bit of VM corresponds to an element of V*j*. Bit 2^{63} corresponds to element 0; bit 2^{0} corresponds to element 63.

The type of test made by the instruction depends on the low-order 2 bits of the *k* designator. The high-order bit of the *k* designator is not interpreted.

If the *k* designator is 0, the VM bit is set to 1 when (V*j* element) is 0 and is set to 0 when (V*j* element) is nonzero.

If the *k* designator is 1, the VM bit is set to 1 when (V*j* element) is nonzero and is set to 0 when (V*j* element) is 0.

If the *k* designator is 2, the VM bit is set to 1 when (V*j* element) is positive and is set to 0 when (V*j* element) is negative. A zero value is considered positive.

If the *k* designator is 3, the VM bit is set to 1 when (V*j* element) is negative and is set to 0 when (V*j* 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 V*j* are zeroed.

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

HOLD ISSUE CONDITIONS: V*j* reserved as operand

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

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

EXECUTION TIME: Instruction issue, 1 CP

V*j* ready, (VL) + 3 CPs if data available†

Except for instruction 073, VM ready (VL) + 4 CPs if data available†

For instruction 073, VM ready (VL) + 5 CPs if data available†

SPECIAL CASES: *k*=0 or 4, VM bit *xx*=1 if (V*j* element *xx*)=0.

k=1 or 5, VM bit *xx*=1 if (V*j* element *xx*)≠0.

k=2 or 6, VM bit *xx*=1 if (V*j* element *xx*) is positive; 0 is a positive condition.

k=3 or 7, VM bit *xx*=1 if (V*j* element *xx*) is negative.

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain starting with that load.

CAL Syntax	Description	Octal Code
V*i* ,A0,A*k*	Transmit (VL) words from memory to V*i* elements starting at memory address (A0) and incrementing by (A*k*) for successive addresses	176*i*0*k*
V*i* ,A0,1†	Transmit (VL) words from memory to V*i* elements starting at memory address (A0) and incrementing by 1 for successive addresses	176*i*00
,A0,A*k* V*j*	Transmit (VL) words from V*j* elements to memory starting at memory address (A0) and incrementing by (A*k*) for successive addresses	1770*jk*
,A0,1 V*j*†	Transmit (VL) words from V*j* elements to memory starting at memory address (A0) and incrementing by 1 for successive addresses	1770*j*0

Instructions 176 and 177 transfer blocks of data between V registers and memory.

Instruction 176 transfers data from memory to elements of register V*i*.

Instruction 177 transfers data from elements of register V*j* to memory.

Register elements begin with 0 and are incremented by 1 for each transfer. Memory addresses begin with (A0) and are incremented by the contents of A*k*. A*k* contains a signed 22-bit integer which is added to the address of the current word to obtain the address of the next word. The 2 high-order bits of (A*k*) are ignored. A*k* can 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: For instruction 176 if Ports A and B busy

For instruction 177 if Port C busy

A0 reserved

A*k* reserved where *k*=1 through 7

† Special CAL syntax

HOLD ISSUE CONDITIONS (continued):

Scalar reference in CP1, CP2, or CP3

For instruction 176, V register *i* reserved as operand or result

For instruction 177, V register *j* reserved as operand

If not bidirectional memory mode, then instruction 176 holds on Port C busy and instruction 177 holds on Port A or B busy.

EXECUTION TIME:

For instruction 176:
Instruction issue, 1 CP
V*i* ready, (VL) + 17 CPs if memory is available
Port A or B busy, (VL) + 5 CPs

For instruction 177:
Instruction issue, 1 CP
V*j* ready, (VL) +3 CPs if data is available
Port C busy, (VL) + 6 CPs

SPECIAL CASES:

Increment (A0)=1 if *k*=0.

Instruction 176 uses Port B. If Port B is busy at issue time, instruction 176 uses Port A. Instruction 177 used Port C.

(A*k*) determines the memory increment. Successive addresses are located in successive banks. References to the same bank can be made every 4 CPs or more. Incrementing (A*k*) by 32 places successive memory references in the same bank, so a word is transferred every 4 CPs or more. If the address is incremented by 16, every other reference is to the same bank, and words can transfer no faster than one every 2 CPs. With any address incrementing that allows 4 CPs before addressing the same bank, the words can transfer each CP.

Memory conflict can slow loading or storing of individual vector elements. The elements are loaded or stored in order, so any delay for any element delays all succeeding elements.

For instruction 176, if there is an instruction using its destination register as a source, the execution of that instruction is delayed whenever there is a delay in instruction 176 results.

APPENDIX SECTION

INSTRUCTION SUMMARY

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
000000	ERR	5-7	-	Error exit
††0010*jk*	CA,A*j* A*k*	5-8	-	Set the channel (A*j*) current address to (A*k*) and begin the I/O sequence
††0011*jk*	CL,A*j* A*k*	5-8	-	Set the channel (A*j*) limit address to (A*k*)
††0012*j*0	CI,A*j*	5-8	-	Clear channel (A*j*) interrupt flag; clear device master-clear (output channel).
††0012*j*1	MC,A*j*	5-8	-	Clear channel (A*j*) interrupt flag; set device master-clear (output channel); clear device ready-held (input channel).
††0013*j*0	XA A*j*	5-8	-	Enter XA register with (A*j*)
††0014*j*0	RT S*j*	5-10	-	Enter RTC register with (S*j*)
††001401	IP 1	5-10	-	Set interprocessor interrupt
††001402	IP 0	5-10	-	Clear interprocessor interrupt
††001403	CLN 0	5-10	-	Enter CLN register with 0
††001413	CLN 1	5-10	-	Enter CLN register with 1
††001423	CLN 2	5-10	-	Enter CLN register with 2
††001433	CLN 3	5-10	-	Enter CLN register with 3
††0014*j*4	PCI S*j*	5-10	-	Enter II register with (S*j*)
††001405	CCI	5-10	-	Clear PCI request
††001406	ECI	5-10	-	Enable PCI request
††001407	DCI	5-10	-	Disable PCI request
00200*k*	VL A*k*	5-12	-	Transmit (A*k*) to VL register
†002000	VL 1	5-12	-	Transmit 1 to VL register
002100	EFI	5-13	-	Enable interrupt on floating-point error
002200	DFI	5-13	-	Disable interrupt on floating-point error
002300	ERI	5-13	-	Enable operand range interrupts

† Special syntax form
†† Privileged to monitor mode

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
002400	DRI	5-13	-	Disable operand range interrupts
002500	DBM	5-13	-	Disable bidirectional memory transfers
002600	EBM	5-13	-	Enable bidirectional memory transfers
002700	CMR	5-13	-	Complete memory references
0030*j*0	VM S*j*	5-15	-	Transmit (S*j*) to VM register
†003000	VM 0	5-15	-	Clear VM register
0034*jk*	SM*jk* 1,TS	5-15	-	Test & set semaphore *jk* in SM
0036*jk*	SM*jk* 0	5-15	-	Clear semaphore *jk* in SM
0037*jk*	SM*jk* 1	5-15	-	Set semaphore *jk* in SM
004000	EX	5-17	-	Normal exit
0050*jk*	J B*jk*	5-18	-	Jump to (B*jk*)
006*ijkm*	J *exp*	5-19	-	Jump to *exp*
007*ijkm*	R *exp*	5-20	-	Return jump to *exp*; set B00 to P.
010*ijkm*	JAZ *exp*	5-21	-	Branch to *exp* if (A0)=0
011*ijkm*	JAN *exp*	5-21	-	Branch to *exp* if (A0)≠0
012*ijkm*	JAP *exp*	5-21	-	Branch to *exp* if (A0) positive; 0 is positive.
013*ijkm*	JAM *exp*	5-21	-	Branch to *exp* if (A0) negative
014*ijkm*	JSZ *exp*	5-23	-	Branch to *exp* if (S0)=0
015*ijkm*	JSN *exp*	5-23	-	Branch to *exp* if (S0)≠0
016*ijkm*	JSP *exp*	5-23	-	Branch to *exp* if (S0) positive; 0 is positive.
017*ijkm*	JSM *exp*	5-23	-	Branch to *exp* if (S0) negative
020*ijkm*	A*i* *exp*	5-25	-	Transmit *exp*=*jkm* to A*i*
021*ijkm*	A*i* *exp*	5-25	-	Transmit *exp*=ones complement of *jkm* to A*i*
022*ijk*	A*i* *exp*	5-26	-	Transmit *exp*=*jk* to A*i*
023*ij*0	A*i* S*j*	5-27	-	Transmit (S*j*) to A*i*
023*i*01	A*i* VL	5-27	-	Transmit (VL) to A*i*
024*ijk*	A*i* B*jk*	5-28	-	Transmit (B*jk*) to A*i*
025*ijk*	B*jk* A*i*	5-28	-	Transmit (A*i*) to B*jk*
026*ij*0	A*i* PS*j*	5-29	Pop/LZ	Population count of (S*j*) to A*i*
026*ij*1	A*i* QS*j*	5-29	Pop/LZ	Population count parity of (S*j*) to A*i*
026*ij*7	A*i* SB*j*	5-29	-	Transmit (SB*j*) to A*i*
027*ij*0	A*i* ZS*j*	5-31	Pop/LZ	Leading zero count of (S*j*) to A*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
027*ij*7	SB*j* A*i*	5-31	-	Transmit (A*i*) to SB*j*
030*ijk*	A*i* A*j*+A*k*	5-32	A Int Add	Integer sum of (A*j*) and (A*k*) to A*i*
†030*i*0*k*	A*i* A*k*	5-32	A Int Add	Transmit (A*k*) to A*i*
†030*ij*0	A*i* A*j*+1	5-32	A Int Add	Integer sum of (A*j*) and 1 to A*i*
031*ijk*	A*i* A*j*-A*k*	5-32	A Int Add	Integer difference of (A*j*) less (A*k*) to A*i*
†031*i*00	A*i* -1	5-32	A Int Add	Transmit -1 to A*i*
†031*i*0*k*	A*i* -A*k*	5-32	A Int Add	Transmit the negative of (A*k*) to A*i*
†031*ij*0	A*i* A*j*-1	5-32	A Int Add	Integer difference of (A*j*) less 1 to A*i*
032*ijk*	A*i* A*j**A*k*	5-33	A Int Mult	Integer product of (A*j*) and (A*k*) to A*i*
033*i*00	A*i* CI	5-34	-	Channel number to A*i* (*j*=0)
033*ij*0	A*i* CA,A*j*	5-34	-	Address of channel (A*j*) to A*i* (*j*≠0; *k*=0)
033*ij*1	A*i* CE,A*j*	5-34	-	Error flag of channel (A*j*) to A*i* (*j*≠0; *k*=1)
034*ijk*	B*jk*,A*i* ,A0	5-36	Memory	Read (A*i*) words to B register *jk* from (A0)
†034*ijk*	B*jk*,A*i* 0,A0	5-36	Memory	Read (A*i*) words to B register *jk* from (A0)
035*ijk*	,A0 B*jk*,A*i*	5-36	Memory	Store (A*i*) words at B register *jk* to (A0)
†035*ijk*	0,A0 B*jk*,A*i*	5-36	Memory	Store (A*i*) words at B register *jk* to (A0)
036*ijk*	T*jk*,A*i* ,A0	5-36	Memory	Read (A*i*) words to T register *jk* from (A0)
†036*ijk*	T*jk*,A*i* 0,A0	5-36	Memory	Read (A*i*) words to T register *jk* from (A0)
037*ijk*	,A0 T*jk*,A*i*	5-36	Memory	Store (A*i*) words at T register *jk* to (A0)
†037*ijk*	0,A0 T*jk*,A*i*	5-36	Memory	Store (A*i*) words at T register *jk* to (A0)
040*ijkm*	S*i* *exp*	5-39	-	Transmit *jkm* to S*i*
041*ijkm*	S*i* *exp*	5-39	-	Transmit *exp*=ones complement of *jkm* to S*i*
042*ijk*	S*i* <*exp*	5-40	S Logical	Form ones mask *exp* bits in S*i* from the right; *jk* field gets 64-*exp*.
†042*ijk*	S*i* #>*exp*	5-40	S Logical	Form zeros mask *exp* bits in S*i* from the left; *jk* field gets 64-*exp*.
†042*i*77	S*i* 1	5-40	S Logical	Enter 1 into S*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
†042*i*00	S*i* -1	5-40	S Logical	Enter -1 into S*i*
043*ijk*	S*i* >*exp*	5-40	S Logical	Form ones mask *exp* bits in S*i* from the left; *jk* field gets *exp*.
†043*ijk*	S*i* #<*exp*	5-40	S Logical	Form zeros mask *exp* bits in S*i* from the right; *jk* field gets 64-*exp*.
†043*i*00	S*i* 0	5-40	S Logical	Clear S*i*
044*ijk*	S*i* S*j*&S*k*	5-41	S Logical	Logical product of (S*j*) and (S*k*) to S*i*
†044*ij*0	S*i* S*j*&SB	5-41	S Logical	Sign bit of (S*j*) to S*i*
†044*ij*0	S*i* SB&S*j*	5-41	S Logical	Sign bit of (S*j*) to S*i* (*j*≠0)
045*ijk*	S*i* #S*k*&S*j*	5-41	S Logical	Logical product of (S*j*) and ones complement of (S*k*) to S*i*
†045*ij*0	S*i* #SB&S*j*	5-41	S Logical	(S*j*) with sign bit cleared to S*i*
046*ijk*	S*i* S*j*\S*k*	5-41	S Logical	Logical difference of (S*j*) and (S*k*) to S*i*
†046*ij*0	S*i* S*j*\SB	5-41	S Logical	Toggle sign bit of S*j*, then enter into S*i*
†046*ij*0	S*i* SB\S*j*	5-41	S Logical	Toggle sign bit of S*j*, then enter into S*i* (*j*≠0)
047*ijk*	S*i* #S*j*\S*k*	5-41	S Logical	Logical equivalence of (S*k*) and (S*j*) to S*i*
†047*i*0*k*	S*i* #S*k*	5-41	S Logical	Transmit ones complement of (S*k*) to S*i*
†047*ij*0	S*i* #S*j*\SB	5-41	S Logical	Logical equivalence of (S*j*) and sign bit to S*i*
†047*ij*0	S*i* #SB\S*j*	5-41	S Logical	Logical equivalence of (S*j*) and sign bit to S*i* (*j*≠0)
†047*i*00	S*i* #SB	5-41	S Logical	Enter ones complement of sign bit into S*i*
050*ijk*	S*i* S*j*!S*i*&S*k*	5-41	S Logical	Logical product of (S*i*) and (S*k*) complement ORed with logical product of (S*j*) and (S*k*) to S*i*
†050*ij*0	S*i* S*j*!S*i*&SB	5-41	S Logical	Scalar merge of (S*i*) and sign bit of (S*j*) to S*i*
051*ijk*	S*i* S*j*!S*k*	5-41	S Logical	Logical sum of (S*j*) and (S*k*) to S*i*
†051*i*0*k*	S*i* S*k*	5-41	S Logical	Transmit (S*k*) to S*i*
†051*ij*0	S*i* S*j*!SB	5-41	S Logical	Logical sum of (S*j*) and sign bit to S*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
†051*ij*0	S*i* SB!S*j*	5-41	S Logical	Logical sum of (S*j*) and sign bit to S*i* (*j*≠0)
†051*i*00	S*i* SB	5-41	S Logical	Enter sign bit into S*i*
052*ijk*	S0 S*i*<*exp*	5-45	S Shift	Shift (S*i*) left *exp*=*jk* places to S0
053*ijk*	S0 S*i*>*exp*	5-45	S Shift	Shift (S*i*) right *exp*=64-*jk* places to S0
054*ijk*	S*i* S*i*<*exp*	5-45	S Shift	Shift (S*i*) left *exp*=*jk* places
055*ijk*	S*i* S*i*>*exp*	5-45	S Shift	Shift (S*i*) right *exp*=64-*jk* places
056*ijk*	S*i* S*i*,S*j*<A*k*	5-46	S Shift	Shift (S*i* and S*j*) left (A*k*) places to S*i*
†056*ij*0	S*i* S*i*,S*j*<1	5-46	S Shift	Shift (S*i* and S*j*) left one place to S*i*
†056*i*0*k*	S*i* S*i*<A*k*	5-46	S Shift	Shift (S*i*) left (A*k*) places to S*i*
057*ijk*	S*i* S*j*,S*i*>*Ak*	5-46	S Shift	Shift (S*j* and S*i*) right (A*k*) places to S*i*
†057*ij*0	S*i* S*j*,S*i*>*1*	5-46	S Shift	Shift (S*j* and S*i*) right one place to S*i*
†057*i*0*k*	S*i* S*i*>*Ak*	5-46	S Shift	Shift (S*i*) right (A*k*) places to S*i*
060*ijk*	S*i* S*j*+S*k*	5-48	S Int Add	Integer sum of (S*j*) and (S*k*) to S*i*
061*ijk*	S*i* S*j*-S*k*	5-48	S Int Add	Integer difference of (S*j*) and (S*k*) to S*i*
†061*i*0*k*	S*i* -S*k*	5-48	S Int Add	Transmit negative of (S*k*) to S*i*
062*ijk*	S*i* S*j*+FS*k*	5-49	Fp Add	Floating-point sum of (S*j*) and (S*k*) to S*i*
†062*i*0*k*	S*i* +FS*k*	5-49	Fp Add	Normalize (S*k*) to S*i*
063*ijk*	S*i* S*j*-FS*k*	5-49	Fp Add	Floating-point difference of (S*j*) and (S*k*) to S*i*
†063*i*0*k*	S*i* -FS*k*	5-49	Fp Add	Transmit normalized negative of (S*k*) to S*i*
064*ijk*	S*i* S*j**FS*k*	5-51	Fp Mult	Floating-point product of (S*j*) and (S*k*) to S*i*
065*ijk*	S*i* S*j**HS*k*	5-51	Fp Mult	Half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*
066*ijk*	S*i* S*j**RS*k*	5-51	Fp Mult	Full-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
067*ijk*	S*i* S*j**IS*k*	5-51	Fp Mult	2-floating-point product of (S*j*) and (S*k*) to S*i*
070*ij*0	S*i* /HS*j*	5-53	Fp Rcpl	Floating-point reciprocal approximation of (S*j*) to S*i*
071*i*0*k*	S*i* A*k*	5-54	-	Transmit (A*k*) to S*i* with no sign extension
071*i*1*k*	S*i* +A*k*	5-54	-	Transmit (A*k*) to S*i* with sign extension
071*i*2*k*	S*i* +FA*k*	5-54	-	Transmit (A*k*) to S*i* as unnormalized floating-point number
071*i*30	S*i* 0.6	5-54	-	Transmit constant 0.75*2**48 to S*i*
071*i*40	S*i* 0.4	5-54	-	Transmit constant 0.5 to S*i*
071*i*50	S*i* 1.	5-54	-	Transmit constant 1.0 to S*i*
071*i*60	S*i* 2.	5-54	-	Transmit constant 2.0 to S*i*
071*i*70	S*i* 4.	5-54	-	Transmit constant 4.0 to S*i*
072*i*00	S*i* RT	5-56	-	Transmit (RTC) to S*i*
072*i*02	S*i* SM	5-56	-	Transmit (SM) to S*i*
072*ij*3	S*i* ST*j*	5-56	-	Transmit (ST*j*) to S*i*
073*i*00	S*i* VM	5-56	-	Transmit (VM) to S*i*
073*ij*1	S*i* SR*j*	5-56	-	Transmit (SR*j*) to S*i* (*j*=0)
073*i*02	SM S*i*	5-56	-	Transmit (S*i*) to SM
073*ij*3	ST*j* S*i*	5-56	-	Transmit (S*i*) to ST*j*
074*ijk*	S*i* T*jk*	5-56	-	Transmit (T*jk*) to S*i*
075*ijk*	T*jk* S*i*	5-56	-	Transmit (S*i*) to T*jk*
076*ijk*	S*i* V*j*,A*k*	5-58	-	Transmit (V*j*, element (A*k*)) to S*i*
077*ijk*	V*i*,A*k* S*j*	5-58	-	Transmit (S*j*) to V*i* element (A*k*)
†077*i*0*k*	V*i*,A*k* 0	5-58	-	Clear V*i* element (A*k*)
10h*ijkm*	A*i* *exp*,A*h*	5-59	Memory	Read from ((A*h*)+*exp*) to A*i* (A0=0)
†100*ijkm*	A*i* *exp*,0	5-59	Memory	Read from (*exp*) to A*i*
†100*ijkm*	A*i* *exp*,	5-59	Memory	Read from (*exp*) to A*i*
†10h*i*00 0	A*i* ,A*h*	5-59	Memory	Read from (A*h*) to A*i*
11h*ijkm*	*exp*,A*h* A*i*	5-59	Memory	Store (A*i*) to (A*h*)+*exp* (A0=0)
†110*ijkm*	*exp*,0 A*i*	5-59	Memory	Store (A*i*) to *exp*
†110*ijkm*	*exp*, A*i*	5-59	Memory	Store (A*i*) to *exp*
†11h*i*00 0	,A*h* A*i*	5-59	Memory	Store (A*i*) to (A*h*)
12h*ijkm*	S*i* *exp*,A*h*	5-59	Memory	Read from ((A*h*)+*exp*) to S*i* (A0=0)

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
†120*ijkm*	S*i* *exp*,0	5-59	Memory	Read from *exp* to S*i*
†120*ijkm*	S*i* *exp*,	5-59	Memory	Read from *exp* to S*i*
†12*hi*00 0	S*i* ,A*h*	5-59	Memory	Read from (A*h*) to S*i*
13*hijkm*	*exp*,A*h* S*i*	5-59	Memory	Store (S*i*) to (A*h*)+*exp* (A0=0)
†130*ijkm*	*exp*,0 S*i*	5-59	Memory	Store (S*i*) to *exp*
†130*ijkm*	*exp*, S*i*	5-59	Memory	Store (S*i*) to *exp*
†13*hi*00 0	,A*h* S*i*	5-59	Memory	Store (S*i*) to (A*h*)
140*ijk*	V*i* S*j*&V*k*	5-61	V Logical	Logical products of (S*j*) and (V*k*) to V*i*
141*ijk*	V*i* V*j*&V*k*	5-61	V Logical	Logical products of (V*j*) and (V*k*) to V*i*
142*ijk*	V*i* S*j*!V*k*	5-61	V Logical	Logical sums of (S*j*) and (V*k*) to V*i*
†142*i*0*k*	V*i* V*k*	5-61	V Logical	Transmit (V*k*) to V*i*
143*ijk*	V*i* V*j*!V*k*	5-61	V Logical	Logical sums of (V*j*) and (V*k*) to V*i*
144*ijk*	V*i* S*j*\V*k*	5-61	V Logical	Logical differences of (S*j*) and (V*k*) to V*i*
145*ijk*	V*i* V*j*\V*k*	5-61	V Logical	Logical differences of (V*j*) and (V*k*) to V*i*
†145*iii*	V*i* 0	5-61	V Logical	Clear V*i*
146*ijk*	V*i* S*j*!V*k*&VM	5-61	V Logical	Transmit (S*j*) if VM bit=1; (V*k*) if VM bit=0 to V*i*.
†146*i*0*k*	V*i* #VM&V*k*	5-61	V Logical	Vector merge of (V*k*) and 0 to V*i*
147*ijk*	V*i* V*j*!V*k*&VM	5-61	V Logical	Transmit (V*j*) if VM bit=1; (V*k*) if VM bit=0 to V*i*.
150*ijk*	V*i* V*j*<A*k*	5-65	V Shift	Shift (V*j*) left (A*k*) places to V*i*
†150*ij*0	V*i* V*j*<1	5-65	V Shift	Shift (V*j*) left one place to V*i*
151*ijk*	V*i* V*j*>*Ak*	5-65	V Shift	Shift (V*j*) right (A*k*) places to V*i*
†151*ij*0	V*i* V*j*>1	5-65	V Shift	Shift (V*j*) right one place to V*i*
152*ijk*	V*i* V*j*,V*j*<A*k*	5-67	V Shift	Double shift (V*j*) left (A*k*) places to V*i*
†152*ij*0	V*i* V*j*,V*j*<1	5-67	V Shift	Double shift (V*j*) left one place to V*i*
153*ijk*	V*i* V*j*,V*j*>A*k*	5-67	V Shift	Double shift (V*j*) right (A*k*) places to V*i*
†153*ij*0	V*i* V*j*,V*j*>1	5-67	V Shift	Double Shift (V*j*) right one place to V*i*

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
154*ijk*	V*i*	S*j*+V*k*	5-72	V Int Add	Integer sums of (S*j*) and (V*k*) to V*i*
155*ijk*	V*i*	V*j*+V*k*	5-72	V Int Add	Integer sums of (V*j*) and (V*k*) to V*i*
156*ijk*	V*i*	S*j*-V*k*	5-72	V Int Add	Integer differences of (S*j*) and (V*k*) to V*i*
†156*i*0*k*	V*i*	-V*k*	5-72	V Int Add	Transmit negative of (V*k*) to V*i*
157*ijk*	V*i*	V*j*-V*k*	5-72	V Int Add	Integer differences of (V*j*) and (V*k*) to V*i*
160*ijk*	V*i*	S*j**FV*k*	5-74	Fp Mult	Floating-point products of (S*j*) and (V*k*) to V*i*
161*ijk*	V*i*	V*j**FV*k*	5-74	Fp Mult	Floating-point products of (V*j*) and (V*k*) to V*i*
162*ijk*	V*i*	S*j**HV*k*	5-74	Fp Mult	Half-precision rounded floating-point products of (S*j*) and (V*k*) to V*i*
163*ijk*	V*i*	V*j**HV*k*	5-74	Fp Mult	Half-precision rounded floating-point products of (V*j*) and (V*k*) to V*i*
164*ijk*	V*i*	S*j**RV*k*	5-74	Fp Mult	Rounded floating-point products of (S*j*) and (V*k*) to V*i*
165*ijk*	V*i*	V*j**RV*k*	5-74	Fp Mult	Rounded floating-point products of (V*j*) and (V*k*) to V*i*
166*ijk*	V*i*	S*j**IV*k*	5-74	Fp Mult	2-floating-point products of (S*j*) and (V*k*) to V*i*
167*ijk*	V*i*	V*j**IV*k*	5-74	Fp Mult	2-floating-point products of (V*j*) and (V*k*) to V*i*
170*ijk*	V*i*	S*j*+FV*k*	5-77	Fp Add	Floating-point sums of (S*j*) and (V*k*) to V*i*
†170*i*0*k*	V*i*	+FV*k*	5-77	Fp Add	Normalize (V*k*) to V*i*
171*ijk*	V*i*	V*j*+FV*k*	5-77	Fp Add	Floating-point sums of (V*j*) and (V*k*) to V*i*
172*ijk*	V*i*	S*j*-FV*k*	5-77	Fp Add	Floating-point differences of (S*j*) and (V*k*) to V*i*
†172*i*0*k*	V*i*	-FV*k*	5-77	Fp Add	Transmit normalized negatives of (V*k*) to V*i*
173*ijk*	V*i*	V*j*-FV*k*	5-77	Fp Add	Floating-point differences of (V*j*) and (V*k*) to V*i*
174*ij*0	V*i*	/HV*j*	5-79	Fp Rcpl	Floating-point reciprocal approximations of (V*j*) to V*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
174*ij*1	V*i* PV*j*	5-80	V Pop	Population counts of (V*j*) to V*i*
174*ij*2	V*i* QV*j*	5-80	V Pop	Population count parities of (V*j*) to V*i*
1750*j*0	VM V*j*,Z	5-82	V Logical	VM=1 where (V*j*)=0
1750*j*1	VM V*j*,N	5-82	V Logical	VM=1 where (V*j*)≠0
1750*j*2	VM V*j*,P	5-82	V Logical	VM=1 if (V*j*) positive; 0 is positive.
1750*j*3	VM V*j*,M	5-82	V Logical	VM=1 if (V*j*) negative
176*i*0*k*	V*i* ,A0,A*k*	5-84	Memory	Read (VL) words to V*i* from (A0) incremented by (A*k*)
†176*i*00	V*i* ,A0,1	5-84	Memory	Read (VL) words to V*i* from (A0) incremented by 1
1770*jk*	,A0,A*k* V*j*	5-84	Memory	Store (VL) words from V*j* to (A0) incremented by (A*k*)
†1770*j*0	,A0,1 V*j*	5-84	Memory	Store (VL) words from V*j* to (A0) incremented by 1

† Special syntax form

6 MBYTES PER SECOND CHANNEL DESCRIPTIONS

B

INTRODUCTION

Each input or output 6 Mbytes per second channel directly accesses Central 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 Central Memory words into 16-bit parcels or 16-bit parcels into 64-bit Central Memory words. Four parcels make up one Central Memory word with bits of the parcels assigned to memory bit positions (see section 2 of this publication).

Each input or output channel has a data channel (4 parity bits, 16 data bits, and 3 control lines), a 64-bit assembly or disassembly register, a channel Current Address (CA) register, and a channel Limit Address (CL) register.

Three control signals (Ready, Resume, and Disconnect) coordinate the transfer of parcels over the channels. In addition to the three control signals, the output channel of the pair has a Master Clear line.

This appendix describes the signal sequence of a 6 Mytes per second input channel and an output channel.

6 MBYTES PER SECOND INPUT CHANNEL SIGNAL SEQUENCE

A general view of a 6 Mbytes per second input channel signal sequence is illustrated in table B-1. The data bits, parity bits, and each signal in the sequence are described below.

DATA BITS 2^0 THROUGH 2^{15}

Data bits 2^0, 2^1, ..., 2^{15} are signals carrying the 16-bit parcel of data from the external device to Central Memory. The data bits must all be valid within 25 nanoseconds after the leading edge of the Ready signal. Data bit signals must remain unchanged on the lines until the corresponding Resume signal is received by the external device. Normally, data is sent coincidentally with the Ready signal and is held until the subsequent Ready signal.

Table B-1. Input channel signal exchange

Central Memory	Channel	External Equipment
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. If (CA)=(CL), go to 13.		
11.		If more data, go to 2.
12.	←	Disconnect (ignored if CA=CL or if channel not active).
13. Set interrupt and deactivate channel.		

† Step 2 can initially precede step 1; that is, the first parcel and ready signal can arrive before requested.

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 follow.

Parity bit	Data bits
0	$2^0 - 2^3$
1	$2^4 - 2^7$
2	$2^8 - 2^{11}$
3	$2^{12} - 2^{15}$

Parity bits are sent from the external device to Central Memory at the same time as data bits and are held stable in the same way as the data bits.

READY SIGNAL

The Ready signal sent to Central Memory indicates a parcel of data is being sent to the Central Memory input channel and can be sampled. A Ready signal is a pulse 50 ±10 nanoseconds wide (at 50% voltage points). The leading edge of the Ready signal at Central Memory begins the timing for sampling the data bits.

RESUME SIGNAL

The Resume signal is sent from Central Memory to the external device showing the parcel was received and Central Memory is ready for the next data transmission. A Resume signal is a pulse 50 ±8 nanoseconds wide (at 50% voltage points).

DISCONNECT SIGNAL

The Disconnect signal is sent from the external device to Central Memory and indicates transmission from the external device is complete. The Disconnect signal is sent after the Resume signal is received for the last Ready signal. A Disconnect signal is a pulse 50 ±10 nanoseconds wide (at the 50% voltage points).

6 MBYTES PER SECOND OUTPUT CHANNEL SIGNAL SEQUENCE

A general view of a 6 Mbytes per second output channel signal sequence is illustrated in table B-2. The data bits, parity bits, and each signal in the sequence are described following the table.

Table B-2. Output channel signal exchange

Central Memory	Channel	External Equipment
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)≠(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 Central Memory to an external device. The data bits are sent concurrently within 5 nanoseconds of the leading edge of the Ready signal. Data bit signals remain steady on the lines until the Resume signal is received.

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 follow:

Parity bit	Data bits
0	$2^0 - 2^3$
1	$2^4 - 2^7$
2	$2^8 - 2^{11}$
3	$2^{12} - 2^{15}$

Parity bits are sent from Central Memory to the external device at the same time as the data bits and are held stable in the same way as the data bits.

READY SIGNAL

The Ready signal sent from Central Memory to the external device indicates data is present and can be sampled. A Ready signal is a pulse 50 $\pm$8 nanoseconds wide (at 50% voltage points). The leading edge of the Ready signal can be used to time data sampling in the external device.

RESUME SIGNAL

The Resume signal is sent from the external device to Central Memory showing the parcel was received and the external device is ready for the next parcel transmission. A Resume signal is a pulse 50 $\pm$10 nanoseconds wide (at 50% voltage points).

DISCONNECT SIGNAL

The Disconnect signal is sent from Central Memory to the external device and indicates transmission from Central Memory is complete. The Disconnect signal is sent after Central Memory receives the Resume signal from the last Ready signal. A Disconnect signal is a pulse 50 $\pm$8 nanoseconds wide (at 50% voltage points).

INDEX

INDEX

6 Mbytes per second channel, 2-12, 2-14, 2-16, B-1
100 Mbytes per second channel, 2-12, 2-14
1250 Mbytes per second channel, 2-12, 2-14

A registers, see Address registers
Addition algorithm, 4-25
Address Add functional unit, 4-4, 4-15
Address functional units, 4-4, 4-14
Address Multiply functional unit, 4-4, 4-15
Address processing, 4-1
Address registers (A), 2-10, 3-12, 4-3
Algorithms
    addition, 4-25
    division, 4-26
    multiplication, 4-25
AND function, 4-33
Arithmetic operations, 4-20
    floating-point, 4-21
    integer, 4-1, 4-20
Auxiliary I/O Processor, 1-17, 1-18, 2-14

B registers, see Intermediate address registers
Beginning address registers, 3-3
BIOP, see Buffer I/O Processor
Branching
    backward, 3-4
    forward, 3-4
Buffer I/O Processor, 1-16, 1-18, 2-14
Buffer Memory, 1-8, 1-18
Buffers, 3-3

CA register, see Current Address register
CAL, see Cray Assembly Language
Central Memory, 1-3, 1-7, 2-1
    16-bank phasing, 2-6
    access, 2-3, 2-20
    access priorities, 2-5
    access time, 2-1, 2-3
    addressing, 2-3
        16 banks, 2-3
        32 banks, 2-3
    bank conflicts, 2-20
    conflict resolution, 2-5
    conflicts, 2-5
        Bank Busy, 2-5
        Section, 2-5
        Simultaneous Bank, 2-5
Central Memory, (continued)
    cycle time, 2-1
    data path with SECDED, 2-6
    error correction/error detection (SECDED), 2-6
    error data, 3-7
    field protection, 3-15
    organization, 2-2
    size, 1-1, 1-3, 1-7, 2-1
    transfer rate, 2-1
Central Processing Unit
    computation section, 1-5, 4-1
    control paths, 1-6
    control section, 1-5, 3-1
    data paths, 1-6
    input/output, 1-17
    input/output section, 1-5, 2-12
    instruction format, 5-1
    instruction summary, A-1
    instructions, 5-1
    inter-CPU communication section, 2-8
    memory section, 2-1
    organization, 1-5, 2-2
    shared resources, 2-1
    speed, 1-2
Chaining, 4-12
Channel bits
    data bits, 2-16, B-1, B-4
    parity bits, 2-16, B-2, B-5
Channel control signals, 2-16, B-3, B-5
Channel groups, 2-20
Channel I/O control, 2-21
Channel Limit register (CL), 2-16
Channel programming
    input, 2-17
    input channel error conditions, 2-18
    multi-CPU, 2-15
    output, 2-19
    output channel error conditions, 2-19
Channel signals
    disconnect, B-3, B-5
    ready, B-3, B-5
    resume, B-3, B-5
Channel word assembly/disassembly, 2-17
Channels
    6 Mbytes per second, 2-12
        instructions, 2-14
        operation, 2-16
        signal descriptions, B-1
        signal sequence, B-1
    100 Mbytes per second, 2-12, 2-14
    1250 Mbytes per second, 2-12, 2-14
    input channel error conditions, 2-18

input channel programming, 2-17
input channel signal sequence, B-1
output channel programming, 2-19
output channel signal sequence, B-3
CIP register, see Current Instruction Parcel register
CL register, see Channel Limit register
Clear programmable clock interrupt request, 3-19
CLN register, see Cluster Number register
Clock
Programmable, 3-18
clear interrupt request, 3-19
instructions, 3-18
Interrupt Countdown counter (ICD), 3-19
Interrupt Interval register (II), 3-18
Real-time, see Real-time Clock register
Clock period, 1-2, 1-4
Cluster Number register (CLN), 3-11
Computation section, 1-5, 4-1
Condensing units, 1-12
Configuration, 1-15
Control section, 3-1
Conventions, 1-4
Clock period, 1-4
Italics, 1-4
Number conventions, 1-4
Register conventions, 1-4
CPU, see Central Processing Unit
Cray Assembly Language (CAL), 5-5
CRAY X-MP Computer System, 1-1
characteristics, 1-2
components, 1-5
configuration, 2-1, 1-15
models, 1-1, 1-15
CSB - read address, 3-7
Current Address register (CA), 2-16
Current Instruction Parcel register (CIP), 3-2

Data Base Address register (DBA), 3-17
Data bits, 2-16, B-1, B-4
Data Limit Address register (DLA), 3-17
Data transfer
I/O Subsystem, 2-14
Solid-state Storage Device, 2-14
DBA register, see Data Base Address register
DCU-4, see Disk controller unit
DD-29, see Disk storage unit
Deadstart, 1-20
sequence, 3-19
Derivation of the division algorithm, 4-28
DIOP, see Disk I/O Processor
Direct memory access ports, 1-18
Disconnect signal, 2-16, B-3, B-5
Disk controller unit (DCU-4), 1-9
Disk I/O Processor, 1-16, 1-18, 2-14
Disk storage unit (DD-29), 1-9, 1-20
Division algorithm, 4-26
DLA register, see Data Limit Address register
DMA, see Direct memory access ports
Double-precision numbers, 4-24
DSU, see Disk storage unit

E - error type, 3-3
Error correction/error detection (SECDED)
description, 2-6
matrix, 2-8
Exchange Address register (XA), 3-8, 4-4
Exchange mechanism, 3-5
Exchange package, 3-5
Active, 3-12
Management, 3-14
Exchange registers, 3-8
Exchange sequence, 3-12
initiate, 3-13
initiated by deadstart sequence, 3-13
initiated by interrupt flag set, 3-13
initiated by program exit, 3-13
issue conditions, 3-14
Exclusive NOR function, 4-33
Exclusive OR function, 4-33

F register, see Flag register
Fetch operations, 2-4
First word address, 2-17
Flag register (F), 3-10
Flags, 3-8
Floating-point Add functional unit, 4-19, 4-22
Floating-point addition, 4-25
Floating-point arithmetic, 4-1, 4-21
Floating-point data format, 4-21
Floating-point functional units, 4-18
Floating-point Multiply functional unit, 4-19, 4-23
Floating-point multiply partial-product sums pyramid, 4-27
Floating-point range errors, 4-22
Floating-point Reciprocal Approximation functional unit, 4-19, 4-24
Floating-point subtraction, 4-25
Front-end computers, 1-19
interfaces, 1-19
Functional units, 4-14
address, 4-1, 4-4, 4-14
Address Add, 4-4, 4-15
Address Multiply, 4-4, 4-15
floating-point, 4-18
Floating-point Add, 4-19, 4-22
Floating-point Multiply, 4-19, 4-23
Reciprocal Approximation, 4-19, 4-24
scalar, 4-1, 4-6, 4-15
Scalar Add, 4-7, 4-15
Scalar Logical, 4-7, 4-16
Scalar Population/Parity/Leading Zero, 4-7, 4-16
Scalar Shift, 4-7, 4-16
vector, 4-1, 4-10, 4-16
Vector Add, 4-10, 4-17
Vector Logical, 4-10, 4-18
Vector Population/Parity, 4-10, 4-18
Vector Shift, 4-10, 4-17
FWA, see First word address

g field, 5-1

h field, 5-1

i field, 5-1
I/O, see Input/output
I/O instructions, 2-15
I/O interrupts, 2-16
I/O lockout, 2-20
I/O memory addressing, 2-23
I/O memory conflicts, 2-23
I/O memory reference, 2-4
I/O memory request conditions, 2-23
I/O Processor, 1-8, 1-16, 1-18, 2-14
I/O program flowchart, 2-18
I/O Subsystem, 1-8
    chassis, 1-8
    data transfer, 2-14
    description, 1-18
    power distribution unit, 1-13
IBA register, see Instruction Base Address register
ICD, see Interrupt Countdown counter
II register, see Interrupt Interval register
ILA register, see Instruction Limit Address register
Inclusive OR function, 4-33
Input channels, 2-17
    error conditions, 2-18
    programming, 2-17
    signal sequence, B-1
Input/output, 1-3
Input/output data paths, 2-22
Input/output section, 2-12
Instruction Base Address register (IBA), 3-16
Instruction buffers, 3-3
    backward branching, 3-4
    forward branching, 3-4
    in-buffer condition, 3-4
    out-of-buffer condition, 3-4
Instruction control, 3-1
Instruction issue, 3-1, 5-5
Instruction Limit Address register (ILA), 3-16
Instruction parcel, 3-2
Instructions, 3-18, 5-1, A-1
    descriptions, 5-5
    fields, 5-1
    formats, 5-1
    functional unit used, A-1
    programmable clock, 3-18
    summary, A-1
Integer arithmetic, 4-1, 4-20
Integer data formats, 4-20
Integer multiply in Floating-point Multiply functional unit, 4-24
Inter-CPU communication, 2-10
Inter-CPU communication section, 1-5, 2-8
Inter-CPU control, 2-10
Interfaces, 1-15, 1-19
Intermediate address registers (B), 3-12, 4-5
Intermediate scalar registers (T), 3-12, 4-9
Interrupt Countdown counter (ICD), 3-19
Interrupt Interval register (II), 3-18
IOP, see I/O Processor
Italics, 1-4

j field, 5-1

k field, 5-1

Last word address, 2-17
LIP register, see Lower Instruction Parcel register
Local Memory, 2-14
Logical operations, 4-32
    AND function, 4-33
    exclusive NOR function, 4-33
    exclusive OR function, 4-33
    inclusive OR function, 4-33
    mask, 4-33
Lower Instruction Parcel register (LIP), 3-3
LWA, see Last word address

m field, 5-2
M register, see Mode register
Machine minimum, 4-23
Mainframe
    chassis, 1-7
    physical characteristics, 1-3
    power distribution unit, 1-13
Mask operation, 4-33
Mass storage, 1-3
Master Clear signal, 2-19
Master I/O Processor, 1-16, 1-18
Memories, 1-3
Memory, see Central Memory
Memory conflicts, see Central Memory
Memory error data fields, 3-7
    error type (E), 3-7
    read address (CSB), 3-7
    read mode (R), 3-7
    syndrome (S), 3-7
    vector not used (VNU), 3-8
Memory field protection, 3-15
Memory section, see Central Memory
MIOP, see Master I/O Processor
Mode register (M), 3-8
Model 22 CRAY X-MP, 1-1, 1-15
Model 24 CRAY X-MP, 1-1, 1-15
Motor-generators, 1-14
Multi-CPU programming, 2-15
Multiple-precision operations, 4-24
Multiplication algorithm, 4-25
    full-precision, 4-26
    half-precision, 4-26

Newton's method, 4-28
Next Instruction Parcel register (NIP), 3-2
NIP register, see Next Instruction Parcel register
Normalized floating-point numbers, 4-22
Number conventions, 1-4

One-parcel instruction format, 5-1
Operand range error, 3-18
Operating registers, 4-3
Out-of-range conditions, 4-23
Output channels, 2-19
    error conditions, 2-19
    programming, 2-19
    signal exchange, B-3
Overflow, 4-22

P register, see Program Address register
Parity bits, 2-16, B-2, B-5
Parity error, 2-18
Phasing, 2-6
PN, see Processor Number
Power distribution units, 1-13
Primary registers, 4-3
Processor Number (PN), 3-7
Program Address register (P), 3-2, 3-12
Program range error, 3-17
Program State register (PS), 3-11
Programmable clock, 3-18
    clear interrupt request, 3-19
    instructions, 3-18
    Interrupt Countdown counter (ICD), 3-19
    Interrupt Interval register (II), 3-18
Programmed Master Clear to external device, 2-19
PS register, see Program State register

R - read mode, 3-7
Ready signal, 2-16, B-3, B-5
Real-time Clock register (RTC), 2-9
Reciprocal Approximation functional unit, 4-19, 4-24
Register conventions, 1-4
Registers, 3-2, 4-3
    Address registers (A), 3-12, 4-3
    Beginning address registers, 3-3
    Channel Limit register (CL), 2-16
    Cluster Number register (CLN), 3-11
    Current Address register (CA), 2-16
    Current Instruction Parcel register (CIP), 3-2
    Data Base Address register (DBA), 3-17
    Data Limit Address register (DLA), 3-17
    Exchange Address register (XA), 3-8, 4-4
    Exchange registers, 3-8
    Flag register (F), 3-10
    Instruction Base Address register (IBA), 3-16
    Instruction Limit Address register (ILA), 3-16
    Intermediate address registers (B), 3-12, 4-5
    Intermediate scalar registers (T), 3-12, 4-9
    Interrupt Interval register (II), 3-18
    Lower Instruction Parcel register (LIP), 3-3
    Mode register (M), 3-8
    Next Instruction Parcel register (NIP), 3-2
    Operating registers, 4-3
    Primary registers, 4-3
    Program Address register (P), 3-2, 3-12
    Program State register (PS), 3-11
    Real-time Clock register (RTC), 2-9
    Scalar registers (S), 3-12, 4-6
    Semaphore registers (SM), 2-11, 4-7
    Shared Address registers (SB), 2-10
    Shared Scalar registers (ST), 2-10, 4-7
    Special register values, 5-4
    Status register, 4-8
    Vector control registers, 4-4, 4-13
    Vector Length register (VL), 4-4, 4-13
    Vector Mask register (VM), 4-13
    Vector registers (V), 4-9
Resume signal, 2-16, B-3, B-5
RTC register, see Real-time Clock register

S - syndrome, 3-7
S registers, see Scalar registers
SB registers, see Shared Address registers
Scalar Add functional unit, 4-7, 4-15
Scalar functional units, 4-1, 4-6, 4-15
Scalar Logical functional unit, 4-7, 4-16
Scalar Population/Parity/Leading Zero functional unit, 4-7, 4-16
Scalar processing, 4-1
Scalar reference, 2-4
Scalar registers (S), 2-10, 3-12, 4-6
Scalar Shift functional unit, 4-7, 4-16
SECDED, see Central Memory
Semaphore registers, 2-11, 4-7
Shared Address registers (SB), 2-10
Shared resources, 2-1
Shared Scalar registers (ST), 2-10, 4-7
SM registers, see Semaphore registers
Solid-state Storage Device (SSD), 1-11
    chassis, 1-11
    configured with CRAY X-MP System, 1-16, 1-17, 1-18
    data transfer, 2-14
    power distribution unit, 1-13
Special register values, 5-4
SSD, see Solid-state Storage Device
ST registers, see Shared Scalar registers
Status register, 4-8
System
    components, 1-1, 1-5
    configuration, 1-15
    deadstart, 1-20
    description, 1-1

T registers, see Intermediate scalar registers

Two-parcel instruction format, 5-2
Twos complement integer arithmetic, 4-20

Unexpected Ready signal, 2-18

V registers, see Vector registers
Vector, 4-1
Vector Add functional unit, 4-10, 4-17
Vector control registers, 4-4, 4-13
Vector functional unit reservation, 4-17
Vector functional units, 4-1, 4-10, 4-16
Vector left double shift, 5-68
Vector Length register (VL), 4-4, 4-13
Vector Logical functional unit, 4-10 4-18
Vector Mask register (VM), 4-13
Vector operation, 4-9, 4-16
Vector Population/Parity functional unit,
  4-10, 4-18
Vector processing, 4-1
Vector registers (V), 4-9
    chaining, 4-12
    conflict, 4-11
    reservations, 4-12
Vector right double shift, 5-69
Vector Shift functional unit, 4-10, 4-17
VL register, see Vector Length register
VM register, see Vector Mask register
VNU - vector not used, 3-8

XA register, see Exchange Address register
XIOP, see Auxiliary I/O Processor

READERS COMMENT FORM

CRAY X-MP Mainframe Reference Manual HR-0032

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.

CUT ALONG THIS LINE

FOLD

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.

FOLD

STAPLE

Cray Research, Inc.
Publications Department
1440 Northland Drive
Mendota Heights, MN 55120
612-452-6650
TLX 298444

CRAY RESEARCH, INC.

CRAY-1® COMPUTER SYSTEMS

CRAY-1 S SERIES
HARDWARE
REFERENCE MANUAL

HR-0808

CRAY RESEARCH, INC.

CRAY-1® COMPUTER SYSTEMS

CRAY-1 S SERIES
HARDWARE
REFERENCE MANUAL

HR-0808

Copyright© 1980, 1981 by CRAY RESEARCH, INC. This manual or parts thereof may not be reproduced in any form without permission of CRAY RESEARCH, INC.

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER HR-0808

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	June, 1980 - Original printing
01	December, 1980 - This change corrects and updates material for the Disk Storage Channel and the Block Multiplexer Channel of the I/O Subsystem. Parts 3 and 4 are affected.
A	June, 1981 - This revision incorporates change packet 01. No other changes have been made.
02	July, 1981 - This change corrects the Central memory sizes and phasing associated with certain configurations. Part 1, section 2 and part 2, section 2 are affected.
B	September, 1981 - Reprint with revision. This printing includes technical corrections and information on the expanded I/O Subsystem. All previous printings are obsolete.
B-01	November, 1981 - Change packet. This packet corrects block multiplexer channel information in part 3, section 7, resulting in partial reorganization of the section. Miscellaneous changes were made in part 2, section 6.

CONTENTS

PREFACE iii

PART 1 - SYSTEM

1. SYSTEM DESCRIPTION 1-1

INTRODUCTION 1-1
SYSTEM COMPONENTS 1-4
Central Processing Unit 1-4
Maintenance Control Unit 1-6
Input/Output Subsystem 1-7
Mass storage 1-7
Condensing units 1-11
Power Distribution Units 1-12
Motor-Generators 1-13

2. SYSTEM CONFIGURATION 2-1

INTRODUCTION 2-1
S/250, S/500, S/1000 Models 2-1
S/1200, S/2200, S/4200 Models 2-2
S/1300, S/2300, S/4300 Models 2-3
S/1400, S/2400, S/4400 Models 2-5
MAINTENANCE CONTROL UNIT 2-8
INTERFACES TO FRONT-END COMPUTER 2-8
SYSTEM OPERATION 2-9
I/O Subsystem communication 2-9
Job flow 2-12
Deadstart 2-13

FIGURES

1-1 Typical CRAY-1 Computer System 1-2
1-2 Central Processing Unit chassis variations 1-5
1-3 Maintenance Control Unit 1-6
1-4 I/O Subsystem 1-8
1-5 DCU-3 Disk Controller Unit 1-10
1-6 DD-29 Disk Storage Unit 1-10
1-7 Condensing Unit 1-11

FIGURES (continued)

1-8 Power Distribution Units 1-12
1-9 Motor-generator equipment 1-13
2-1 Block diagram of S/250, S/500, and S/1000 systems 2-1
2-2 Block diagram of S/1200, S/2200, and S4200 systems 2-2
2-3 Block diagram of S/1300, S/2300, S/4300 systems with block multiplexer channels 2-4
2-4 Block diagram of S/1300, S/2300/, and S/4300 systems with increased disk capacity 2-5
2-5 Block diagram of S/1400, S/2400, and S/4400 systems with block multiplexer channels 2-6
2-6 Block diagram of S/1400, S/2400, and S/4400 systems with increased disk capacity 2-7
2-7 I/O Subsystem communication 2-11
2-8 Job flow diagram 2-12

TABLES

1-1 Models of the CRAY-1 S Series of Computer Systems 1-1
1-2 CRAY-1 System characteristics 1-3
1-3 Characteristics of a DD-29 Disk Storage Unit 1-9

PART 2 - CENTRAL PROCESSING UNIT

1. GENERAL INFORMATION 1-1

INTRODUCTION 1-1
Register conventions 1-2
Number conventions 1-2
Clock period 1-2
MEMORY SECTION 1-4
CONTROL SECTION 1-4
COMPUTATION SECTION 1-5
INPUT/OUTPUT SECTION 1-7

2. CENTRAL MEMORY SECTION 2-1

INTRODUCTION 2-1
MEMORY CYCLE TIME 2-1
MEMORY ACCESS 2-1
MEMORY ORGANIZATION 2-3
MEMORY ADDRESSING 2-4
SPEED CONTROL 2-4
8-BANK PHASING 2-5
MEMORY ERROR CORRECTION 2-5

3. CPU CONTROL SECTION 3-1

INSTRUCTION ISSUE AND CONTROL 3-1
P register 3-2
NIP register 3-2
CIP register 3-2
LIP register 3-3
Instruction buffers 3-3
EXCHANGE MECHANISM 3-5
Exchange package 3-5
Memory error data 3-7
Exchange registers 3-7
XA register 3-8
M register 3-8
F register 3-8
Active exchange package 3-9
Exchange sequence 3-9
Initiated by deadstart sequence 3-10
Initiated by interrupt flag set 3-10
Initiated by program exit 3-10
Exchange sequence issue conditions 3-11
Exchange package management 3-11
MEMORY FIELD PROTECTION 3-12
BA register 3-13
LA register 3-13
Program range error 3-13
Operand range error 3-14
REAL-TIME CLOCK 3-14
PROGRAMMABLE CLOCK 3-14
Instructions 3-14
Interrupt interval register 3-15
Interrupt countdown counter 3-15
Clear programmable clock interrupt request 3-15
DEADSTART SEQUENCE 3-16

4. CPU COMPUTATION SECTION 4-1

INTRODUCTION 4-1
OPERATING REGISTERS 4-1
ADDRESS REGISTERS 4-2
A registers 4-3
B registers 4-4
SCALAR REGISTERS 4-5
S registers 4-5
T registers 4-6
VECTOR REGISTERS 4-7
V registers 4-7
V register reservations 4-9
Vector control registers 4-10
VL register 4-10
VM register 4-10

FUNCTIONAL UNITS 4-11
Address functional units 4-12
Address add unit 4-12
Address multiply unit 4-12
Scalar functional units 4-12
Scalar add unit 4-13
Scalar shift unit 4-13
Scalar logical unit 4-13
Scalar population/parity/leading zero unit 4-13
Vector functional units 4-14
Vector functional unit reservation 4-14
Vector add unit 4-14
Vector shift unit 4-15
Vector logical unit 4-15
Vector population/parity unit 4-15
Recursive characteristics of vector functional units 4-16
Floating-point functional units 4-18
Floating-point add unit 4-18
Floating-point multiply unit 4-18
Reciprocal approximation unit 4-19
ARITHMETIC OPERATIONS 4-19
Integer arithmetic 4-19
Floating-point arithmetic 4-20
Normalized floating-point numbers 4-21
Floating-point range errors 4-21
Double-precision numbers 4-23
Addition algorithm 4-23
Multiplication algorithm 4-23
Division algorithm 4-25
Derivation of the CRAY-1 Algorithm 4-26
LOGICAL OPERATIONS 4-31

5. CPU INPUT/OUTPUT SECTION 5-1

INTRODUCTION 5-1
MEMORY CHANNEL 5-1
I/O CHANNELS 5-2
Channel groups 5-2
I/O instructions 5-2
Basic I/O channel operation 5-3
Input channel programming 5-4
Input channel error conditions 5-6
Output channel programming 5-7
Output channel error condition 5-7
16-bit asynchronous channels 5-7
Input channels 5-7
Output channels 5-9

I/O CHANNELS (continued)
16-bit high-speed asynchronous channels 5-11
Input channels 5-11
Output channels 5-13
16-bit synchronous channels 5-15
Input channels 5-15
Output channels 5-17
Programmed master clear to external device 5-19
Sequence for asynchronous channels 5-19
Sequence for high-speed asynchronous and synchronous channels 5-20
Memory access 5-22
I/O lockout 5-22
Memory bank conflicts 5-22
I/O Memory conflicts 5-23
I/O Memory request conditions 5-23
I/O Memory addressing 5-23

6. CPU INSTRUCTIONS 6-1

INSTRUCTION FORMAT 6-1
Arithmetic, logical format 6-1
Shift, mask format 6-2
Immediate constant format 6-2
Memory transfer format 6-3
Branch format 6-4
SPECIAL REGISTER VALUES 6-5
INSTRUCTION ISSUE 6-5
INSTRUCTION DESCRIPTIONS 6-6

7. CPU INTERFACES 7-1

INTRODUCTION 7-1
PHYSICAL DESCRIPTION 7-1
CABLING LIMITATIONS 7-2
OPERATION 7-2

FIGURES

1-1 Basic organization of the CPU 1-1
1-2 Control and data paths in the CPU 1-3
2-1 Memory address (16 banks) 2-4
2-2 Memory address (8 banks) 2-4
2-3 Memory data path with SECDED 2-5
2-4 Error correction matrix 2-7
3-1 Instruction issue and control elements 3-1
3-2 Instruction buffers 3-3
3-3 Exchange package 3-6

FIGURES (continued)

4-1 Address registers and functional units 4-2
4-2 Scalar registers and functional units 4-5
4-3 Vector registers and functional units 4-7
4-4 Integer data formats 4-19
4-5 Floating-point data formats 4-20
4-6 Integer multiply in floating-point multiply unit 4-22
4-7 49-bit floating-point addition 4-23
4-8 Floating-point multiply partial-product sums pyramid 4-24
4-9 Newton's method 4-25
5-1 Basic I/O program flow chart 5-5
5-2 Channel I/O control 5-21
6-1 General form for instructions 6-1
6-2 Format for arithmetic and logical instructions 6-2
6-3 Format for shift and mask instructions 6-2
6-4 Format for immediate constant instructions 6-3
6-5 Format for memory transfer instructions 6-4
6-6 Two-parcel format for branch instructions 6-4
6-7 Vector left double shift, first element, VL greater than 1 . 6-66
6-8 Vector left double shift, second element,VL greater than 2 . 6-66
6-9 Vector left double shift, last element 6-66
6-10 Vector right double shift, first element 6-67
6-11 Vector right double shift, second element, VL greater than 1 6-68
6-12 Vector right double shift, last operation 6-68
7-1 Typical interface cabinet 7-2

TABLES

1-1 Characteristics of the CPU memory section 1-4
1-2 Characteristics of the CPU control section 1-5
1-3 Characteristics of CPU computation section 1-6
1-4 Characteristics of the CPU input/output section 1-7
2-1 Vector memory rate x 80 x 10^6 references per second 2-4
5-1 Channel word assembly/disassembly 5-3
5-2 16-bit asynchronous input channel signal exchange 5-8
5-3 16-bit asynchronous output channel signal exchange 5-10
5-4 16-bit high-speed asynchronous input channel signal exchange 5-12
5-5 16-bit high-speed asynchronous output channel signal exchange 5-14
5-6 16-bit synchronous input channel signal exchange 5-16
5-7 16-bit synchronous output channel signal exchange 5-18

PART 3 - I/O SUBSYSTEM

1. GENERAL INFORMATION 1-1

INTRODUCTION 1-1
MEMORY SECTION 1-1
CONTROL SECTION 1-3
COMPUTATION SECTION 1-3
INPUT/OUTPUT SECTION 1-4
I/O SUBSYSTEM CLOCK 1-4

2. I/O MEMORY SECTION 2-1

INTRODUCTION 2-1
MEMORY SPEEDS 2-1
MEMORY ORGANIZATION 2-2
MEMORY ACCESS 2-2
MEMORY ADDRESSING 2-2
MEMORY PARITY PROTECTION 2-3

3. IOP CONTROL SECTION 3-1

INTRODUCTION 3-1
INSTRUCTION CONTROL NETWORK 3-1
Instruction stack 3-3
Forward relative branch 3-5
Backward relative branch 3-5
II register 3-6
B register 3-6
RP register 3-6
DP register 3-6
P register 3-7
Program exit stack 3-7
Program Exit Stack and I/O Interrupts 3-10
Program Exit Stack Timing Note 3-10
Program fetch request flag 3-10
MA register 3-11

4. IOP COMPUTATION SECTION 4-1

INTRODUCTION 4-1
OPERAND REGISTERS 4-1
FUNCTIONAL UNITS 4-2
Adder 4-2
Shifter 4-2
ACCUMULATOR 4-3
Carry-bit register 4-3
Addend registers 4-4

5. INPUT/OUTPUT SECTION 5-1

INTRODUCTION 5-1
I/O CONFIGURATION 5-1
I/O SPEEDS 5-2
CHANNEL CHARACTERISTICS 5-2
Accumulator channels 5-2
Function designators 5-3
Function strobe 5-3
Accumulator data 5-3
Read done 5-3
Read busy 5-3
Busy/done 5-4
Master clear 5-4
Clock 5-4
Interrupt 5-4
Channels using a DMA port 5-4
I/O Memory data 5-5
I/O Memory address 5-5
Request read 5-5
Request write 5-5
Acknowledge read 5-5
Acknowledge write 5-6
Read sequence 5-6
Write sequence 5-6
STANDARD CHANNELS 5-7
Channel for I/O requests (CH. 0) 5-9
Channel for program fetch request (CH. 1) 5-9
Channel to program exit stack (CH. 2) 5-10
Deadstart sequence 5-11
Channel for I/O Memory error (CH. 3) 5-12
Channel to real-time clock (CH. 4) 5-12
Channel to Buffer Memory (CH. 5) 5-13
Error handling 5-16
Buffer Memory interface deadstart 5-16
Buffer Memory interface dead dump 5-16
CHANNEL FOR I/O PROCESSOR INPUT (CH. 6, 10, 12) 5-16
CHANNEL FOR I/O PROCESSOR OUTPUT (CH. 7, 11, 13) 5-17
INTERRUPT SEQUENCE 5-18

6. IOP INSTRUCTIONS 6-1

INSTRUCTION FORMAT 6-1
INSTRUCTION DESCRIPTIONS 6-2

7. INTERFACES 7-1

INTRODUCTION 7-1
INTERFACE CHARACTERISTICS 7-1
INTERFACE FUNCTION CODES 7-2
DISK STORAGE UNIT CHANNEL 7-6
Disk storage unit characteristics 7-6
DSU data sequence pattern 7-7
Sector identification word 7-8
I/O Memory address register 7-8
Status response register 7-9
DKA : 0 - clear channel 7-9
DKA : 1 - select mode 7-9
Parameter 000xxx - release unit 7-10
Parameter 001xxx - reserve unit 7-10
Parameter 002xxx - clear fault flags 7-11
Parameter 003xxx - return to zero cylinder 7-11
Parameter 004xxx - select cylinder margin 7-11
Parameter 005xxx - read sector number 7-11
Parameter 006xxx - read error flags 7-12
Parameter 007000 - read cylinder register 7-15
Parameter 007001 - read head register 7-15
Parameter 007002 - read margin/difference register 7-16
Parameter 007003 - read interlock register 7-17
Timing notes 7-17
DKA : 2 - read disk data 7-18
DKA : 2 - abnormal conditions 7-19
DKA : 2 - special modes 7-20
Buffer echo mode 7-21
DKA : 3 - write disk data 7-21
Fire code generation 7-22
Lost data error 7-22
Lost function error 7-22
Format mode 7-23
Buffer echo mode 7-23
DKA : 4 - select head group 7-24
DKA : 5 - select cylinder 7-24
DKA : 6 - clear interrupt enable 7-25
DKA : 7 - set interrupt enable 7-25
DKA : 10 - read local memory address 7-25
DKA : 11 - read status response 7-25
DKA : 14 - enter local memory address 7-26
DKA : 15 - status response register diagnostic 7-26
CONSOLE DISPLAY CHANNEL 7-26
CONSOLE KEYBOARD CHANNEL 7-27
PERIPHERAL EXPANDER CHANNEL 7-28
Interface registers 7-28
Channel assignments 7-29
EXB : 0 - idle channel 7-29

PERIPHERAL EXPANDER CHANNEL (continued)
EXB : 1 - DIA 7-29
EXB : 2 - DIB 7-29
EXB : 3 - DIC 7-29
EXB : 4 - read busy/done, interrupt number 7-29
EXB : 5 - load device address 7-30
EXB : 6 - MSKO mask out 7-30
EXB : 7 - set interrupt mode 7-32
EXB : 10 - read data bus status 7-32
EXB : 11 - read status 1 7-32
EXB : 13 - read status 2 7-33
EXB : 14 - DOA (Data out A) 7-34
EXB : 15 - DOB (Data out B) 7-35
EXB : 16 - DOC (Data out C) 7-35
EXB : 17 - send control 7-35
Delayed functions 7-36
Transfer speeds 7-36
CHANNEL FOR INPUT FROM CRAY-1 CHANNEL 7-37
I/O Memory address register 7-37
CIA : 0 - clear channel 7-37
CIA : 1 - enter I/O Memory address 7-38
CIA : 2 - enter parcel count 7-38
CIA : 3 - clear channel parity error flags 7-38
CIA : 4 - clear ready waiting 7-38
CIA : 6 - clear interrupt enable flag 7-38
CIA : 7 - set interrupt enable flag 7-38
CIA : 10 - read memory address 7-38
CIA : 11 - read ready waiting/error flags 7-38
CHANNEL FOR OUTPUT TO CRAY-1 CHANNEL 7-40
I/O Memory address register 7-40
COA : 0 - clear channel 7-40
COA : 1 - enter I/O Memory address 7-41
COA : 2 - enter parcel count 7-41
COA : 3 - clear error flag 7-41
COA : 4 - set/clear external control signals 7-41
COA : 6 - clear interrupt enable flag 7-42
COA : 7 - set interrupt enable flag 7-42
COA : 10 - read I/O Memory address 7-42
COA : 11 - read error flags 7-43
MEMORY CHANNEL 7-43
Signal Descriptions 7-44
Central processing unit to I/O processor input channel 7-44
I/O Processor to central processing unit
output channel 7-48
Memory Channel functions for input from CPU 7-50
Interface registers 7-51
HIA : 0 - clear channel busy, done flags 7-51
HIA : 1 - enter I/O Memory starting address 7-51
HIA : 2 - enter upper Central Memory address 7-51

Memory Channel functions for input from CPU (continued)
HIA : 3 - enter lower Central Memory address 7-51
HIA : 4 - read Central Memory, enter block length . 7-52
HIA : 6 - clear interrupt enable 7-52
HIA : 7 - set interrupt enable 7-52
HIA : 14 - enter diagnostic mode 7-52
Memory channel input error processing 7-53
Memory channel input sequence 7-53
Memory Channel functions for output to CPU 7-56
Interface registers 7-56
HOA : 0 - clear channel busy, done flags 7-56
HOA : 1 - enter I/O Memory address 7-56
HOA : 2 - enter upper Central Memory address 7-57
HOA : 3 - enter lower Central Memory address 7-57
HOA : 5 - write Central Memory, enter block length . 7-57
HOA : 6 - clear interrupt enable 7-57
HOA : 7 - set interrupt enable 7-57
HOA : 14 - enter diagnostic mode 7-57
Central Memory output error processing 7-58
Memory channel output sequence 7-60
ERROR LOGGING CHANNEL 7-61
Interface Registers 7-61
ERA : 0 - idle channel 7-61
ERA : 6 - clear interrupt enable flag 7-62
ERA : 7 - set interrupt enable flag 7-62
ERA : 10 - read error status 7-62
ERA : 11 - read error information (first parameter) . . . 7-63
ERA : 12 - read error information (second parameter) . . 7-63
ERA : 13 - read error information (third parameter) . . . 7-65
BLOCK MULTIPLEXER CHANNEL 7-66
General characteristics 7-66
Transfers rates 7-67
Data handling 7-67
Record size 7-67
Parity 7-68
Interrupts 7-68
BMA : 0 - clear channel busy and done flags 7-69
BMA : 1 - send reset function 7-69
Parameter xxxxx0 - clear output tag lines 7-69
Parameter xxxxx1 - interface disconnect 7-69
Parameter xxxxx2 - selective reset 7-70
Parameter xxxxx3 - system reset 7-70
BMA : 2 - channel command 7-70
Parameter command bits 7-71
BMA : 3 - read request-in address 7-73
BMA : 4 - asynchronous I/O 7-74

BLOCK MULTIPLEXER CHANNEL (continued)
BMA : 5 - delay counter diagnostic 7-76
BMA : 6 - clear channel interrupt enable flag 7-76
BMA : 7 - set channel interrupt enable flag 7-76
BMA : 10 - read I/O Memory address 7-76
BMA : 11 - read byte counter 7-77
BMA : 12 - read status 7-78
BMA : 13 - read input tags 7-79
BMA : 14 - enter I/O Memory address 7-80
BMA : 15 - enter byte count 7-81
BMA : 16 - enter device address/mode 7-81
Parameter mode bits 7-82
BMA : 17 - enter output tags. 7-84
PROGRAMMING EXAMPLES 7-84

8. BUFFER MEMORY 8-1

INTRODUCTION 8-1
MEMORY SPEEDS 8-1
MEMORY ORGANIZATION 8-1
MEMORY ACCESS 8-2
MEMORY ADDRESSING 8-2
ERROR PROTECTION 8-3

FIGURES

1-1 Basic organization of an I/O Processor 1-2
2-1 I/O Memory address format 2-3
3-1 I/O Processor block diagram 3-2
3-2 Instruction format 3-3
3-3 Instruction stack operation 3-4
3-4 Program exit stack 3-8
5-1 Buffer Memory address formation. 5-14
6-1 Instruction format 6-1
7-1 DSU data sequence pattern 7-7
7-2 Sector ID format 7-8
7-3 Status response error flags. 7-13
7-4 Offset margin status word. 7-16
7-5 Difference register example. 7-16
7-6 Interlock register status bits 7-17
7-7 Format mode sector pattern 7-23
7-8 Memory Channel signals 7-44
7-9 Address and word count formats 7-47
7-10 Memory Channel sequence, input to I/O Processor. 7-55
7-11 Memory Channel sequence, output from I/O Processor 7-60
7-12 BMC-4 data assembly/disassembly 7-68
7-13 Channel read sequence. 7-73
7-14 Channel ASYNCHRONOUS I/O sequence. 7-74

FIGURES (continued)

7-15 Asynchronous data and status processing. 7-75
8-1 Parcel packing in memory word. 8-2
8-2 Buffer Memory port assignments 8-2
8-3 Buffer Memory address formation. 8-3

TABLES

2-1 I/O Processor memory characteristics 2-1
3-1 Characteristics of the IOP control section 3-1
4-1 Characteristics of the I/O computation section 4-1
4-2 Accumulator sources and destinations 4-3
5-1 Characteristics of the IOP input/output section. 5-1
5-2 I/O Processor standard channel assignments 5-7
5-3 Standard channel functions 5-8
7-1 Interface functions. 7-3
7-2 Sector ID parity bit assignments 7-9
7-3 DKA : 1 parameters 7-10
7-4 Parameter 006 error flags. 7-14
7-5 Interlock status bits. 7-18
7-6 Peripheral device mask bits for interrupt disabling. 7-31
7-7 Read status 1 bit assignments. 7-33
7-8 Read status 2 bit assignments. 7-34
7-9 Accumulator bit control signals. 7-35
7-10 Ready waiting/error flags. 7-39
7-11 External control signal bits 7-42
7-12 Error flags. 7-43
7-13 Input channel diagnostic modes 7-53
7-14 Input channel error codes. 7-54
7-15 Output channel diagnostic modes. 7-58
7-16 Output channel error codes 7-59
7-17 Error status register bits 7-62
7-18 First error parameter selection. 7-64
7-19 Send reset function parameters 7-69
7-20 Channel command function parameter bits. 7-71
7-21 Channel command bit assignments 7-72
7-22 Read I/O Memory address response bits 7-77
7-23 Status register bits 7-79
7-24 Input tags status bits 7-80
7-25 I/O Memory address register bits 7-81
7-26 Device address register bits 7-82
7-27 Command chaining mode selection 7-83
7-28 Interrupt mode selection 7-83
7-29 Channel type mode selection 7-83
7-30 Output tags register bits. 7-84

PART 4 - APPENDIX SECTION

A. SUMMARY OF CPU TIMING INFORMATION. A-1

SCALAR INSTRUCTIONS. A-1
VECTOR INSTRUCTIONS. A-3
HOLD ISSUE A-4
HOLD MEMORY. A-5
INTERRUPT TIMING A-6

B. PHYSICAL ORGANIZATION OF CPU B-1

MAINFRAME. B-1
Modules B-1
Clock B-4
Power Supplies B-5
COOLING. B-5

C. SOFTWARE CONSIDERATIONS C-1

SYSTEM MONITOR C-1
USER PROGRAM C-1
OPERATING SYSTEM C-1
SYSTEM OPERATION C-2
FLOATING-POINT RANGE ERRORS. C-2

D. CPU INSTRUCTION SUMMARY. D-1

E. I/O PROCESSOR INSTRUCTION SUMMARY. E-1

F. SYSTEM CHANNEL ASSIGNMENTS F-1

G. IOP PROGRAMMING CONSIDERATIONS G-1
EXIT STACK TIMING. G-1
EXIT STACK INTERRUPT HANDLING. G-1
SYSTEM INTERRUPT ENABLE. G-1
SYSTEM INTERRUPT DISABLE G-2
SYSTEM INTERRUPT CLEARED OR SET BY THE ENABLES FOR INDIVIDUAL CHANNELS G-2
I/O CHANNEL TIMING G-2
Buffer memory errors G-2
BUFFER MEMORY DEADSTART TIME G-3
ERROR LOGGING AND BLOCK MULTIPLEXER CHANNELS G-3
I/O INSTRUCTIONS AFTER DEADSTART G-3
PERIPHERAL EXPANDER CHANNEL TRANSFERS. G-3

H. LIST OF ABBREVIATIONS. H-1

FIGURES

B-1 Physical organization of CPU. B-2
B-2 General chassis layout. B-3
B-3 Clock pulse waveform. B-4

TABLES

F-1 Typical Model 4400 system channel assignments F-1

PART 1

SYSTEM

SYSTEM DESCRIPTION 1

INTRODUCTION

The CRAY-1 S Series of Computer Systems is based on a powerful general-purpose central processing unit CPU capable of extremely high processing rates. These rates are achieved by combining scalar and vector capabilities into the CPU, which is joined to a large, fast, bipolar integrated circuit memory. Vector processing, which is the performance of iterative operations on sets of ordered data, provides results at rates greatly exceeding the result rates of conventional scalar processing. Scalar operations complement the vector capability by providing solutions to problems not readily adaptable to vector techniques. Table 1-1 summarizes the models available in the S Series of computer systems. These models are described in greater detail in section 2 under System Configurations. The Model S/250 is no longer available.

Table 1-1. Models of the CRAY-1 S Series of Computer Systems

MODEL	S/250	S/500	S/1000	S/1200	S/1300	S/1400	S/2200	S/2300	S/2400	S/4200	S/4300	S/4400
CPU Central Memory size (64-bit words)	1/4M	1/2M	1M	1M	1M	1M	2M	2M	2M	4M	4M	4M
FRONT-END INTERFACES	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3
I/O SUBSYSTEM I/O Processors				2	3	4	2	3	4	2	3	4
Buffer memory Size				1-8M	1-8M	1-8M	1-8M	1-8M	1-8M	1-8M	1-8M	1-8M
DCU-4 Disk Controller Units				1-4	1-8	1-12	1-4	1-8	1-12	1-4	1-8	1-12
DD-29 Disk Storage Units				2-16	2-32	2-48	2-16	2-32	2-48	2-16	2-32	2-48
Block Multiplexer Controllers					1-4	1-4		1-4	1-4		1-4	1-16
Block Multiplexer Channels					1-16	1-16		1-16	1-16		1-16	1-16
MASS STORAGE SUBSYSTEM DCU-3 Disk Control Units	2-8	2-8	2-8									
DD-29 Disk Storage Units	2-32	2-32	2-32									

The CRAY-1 S Series of Computer Systems encompasses a wide variety of configurations. On the advanced models, a sophisticated I/O Subsystem matches the high processing rates with high input/output transfer rates for communication with mass storage units, other peripheral devices, and a wide variety of host computers. Several combinations of memory size and I/O capabilities are offered.

An optimum system can be configured for a particular use. This section briefly describes the system components, configurations, and operation. Table 1-2 gives the overall system characteristics. Figure 1-1 illustrates a typical system.

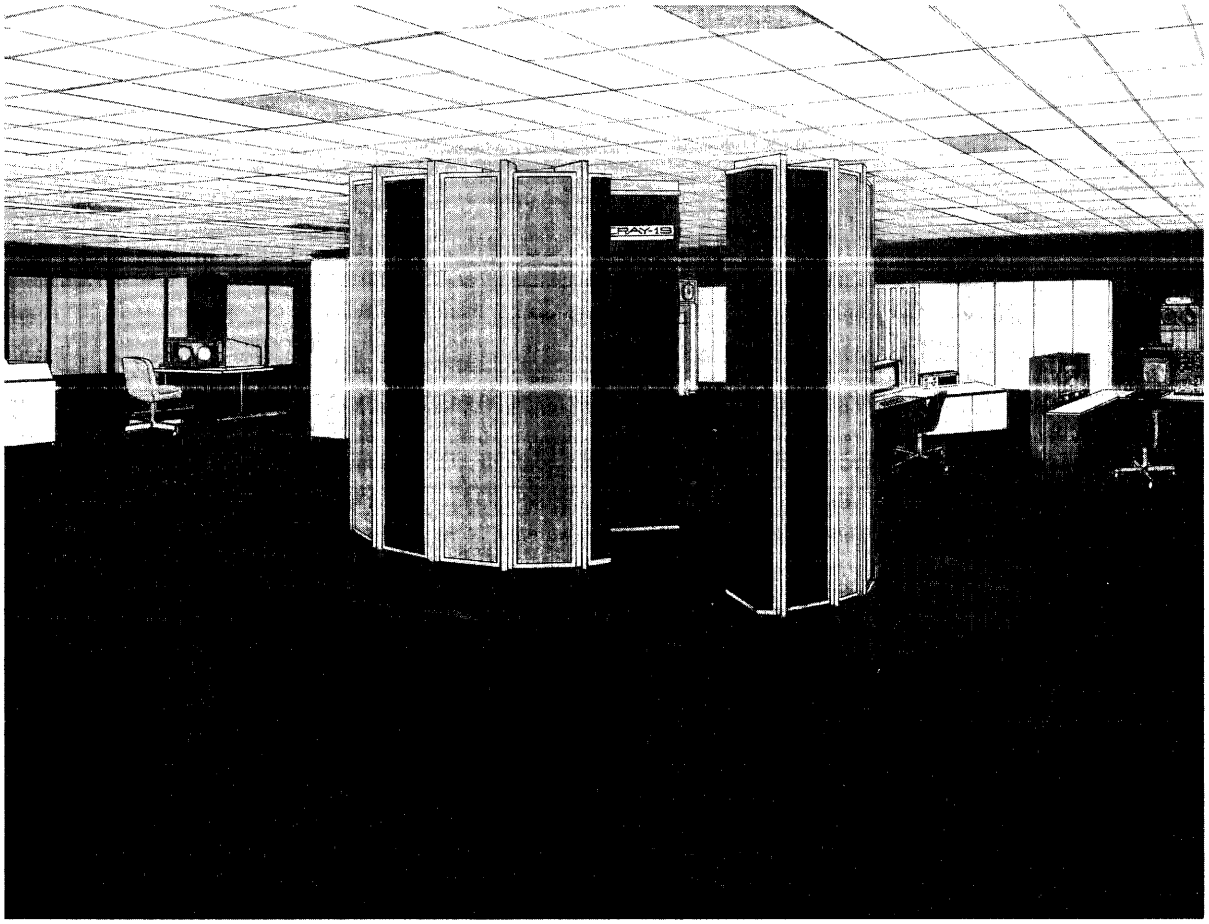


Figure 1-1. Typical CRAY-1 Computer System

Table 1-2. CRAY-1 System characteristics

Configuration	- Central Processing Unit - Maintenance Control Unit (MCU) or I/O Subsystem with 2 to 4 I/O Processors
CPU Speed	- 12.5 ns CPU clock period - 80 million floating-point additions per second rate - 80 million floating-point multiplications per second rate - Simultaneous floating-point addition and multiplication - 80 million 1/2 precision floating-point divisions per second rate - 25 million full precision floating-point divisions per second rate
Memories	- Up to 4 million 64-bit words in CPU Central Memory - 65 thousand 16-bit parcels in each I/O Processor I/O Memory - 1 to 8 million 64-bit words of I/O Subsystem Buffer Memory
Mass Storage	- 600 million byte disk drive - 48 disk drives maximum - 38.7 Mbits/s disk drive transfer rate
Input/Output	- Up to 12 CPU channel pairs (if no I/O Subsystem) - 1 standard and 1 optional Memory Channel to CPU; approx. 800 Mbits/s - Many other mainframe interfaces - 6 Direct memory access (DMA) ports to each I/O Processor - 40 channels; input or output sharing the six DMA ports per I/O Processor
Physical	- 100 sq ft floor space for CPU - 10 sq ft floor space for I/O Subsystem - 5.25 tons, CPU weight - 1.5 tons, I/O Subsystem weight - Liquid refrigeration of each chassis - 400 Hz power from motor-generators

SYSTEM COMPONENTS

The CRAY-1 Computer System is composed of a Central Processing Unit (CPU), a Maintenance Control Unit or I/O Subsystem, and mass storage devices. Supporting this equipment are condensing units for refrigeration, power distribution units for the CPU and the I/O Subsystem, and motor-generators providing system power.

CENTRAL PROCESSING UNIT

The CPU is a single integrated processing unit having a memory section, a control section, a computation section, and an input/output section.

The memory section contains from 262,144 to 4,194,304 words of 64 bits each. Memory is organized in 8 or 16 interleaved banks to allow fast access to successive addresses. The bipolar integrated circuits and the interleaving give a 4-clock-period (50 nanosecond) bank cycle time and an overall memory performance of one word every 12.5 nanoseconds. Single error correction/double error detection (SECDED) protects data integrity.

The control section contains registers and circuitry that controls instruction issue, transfers execution from program to program, and recognizes modes, error flags, and interrupt conditions.

The computation section handles the scalar and vector operations. It is composed of scalar, vector, and address registers; instruction buffers; and segmented functional units.

The input/output (I/O) section contains up to 12 channel pairs; each input or output channel carries 16 data bits, 3 control bits, and 4 parity bits. Normal channel speed is 50 or 160 Mbits per second, maximum. A single Memory channel communicates with the I/O Subsystem at approximately 850 Mbits per second.

Because superior performance is achieved using the Memory channel and I/O Subsystem channels, all but one of the CPU channels are generally not used when an I/O Subsystem is present.

The Central Processing Unit computation section is located in four columns of mainframe chassis. An additional four or eight columns contain Central Memory. Some versions of the CRAY-1 Computer System use the 8-column chassis with four columns of memory and four columns of computation section. Other models have an additional four columns to house more memory. These variations are discussed later with the description of each configuration. The bench at the base of each column houses the DC power supplies for that column. Figure 1-2 shows these variations of CPU chassis. Part 2 of this publication describes the CPU in detail.

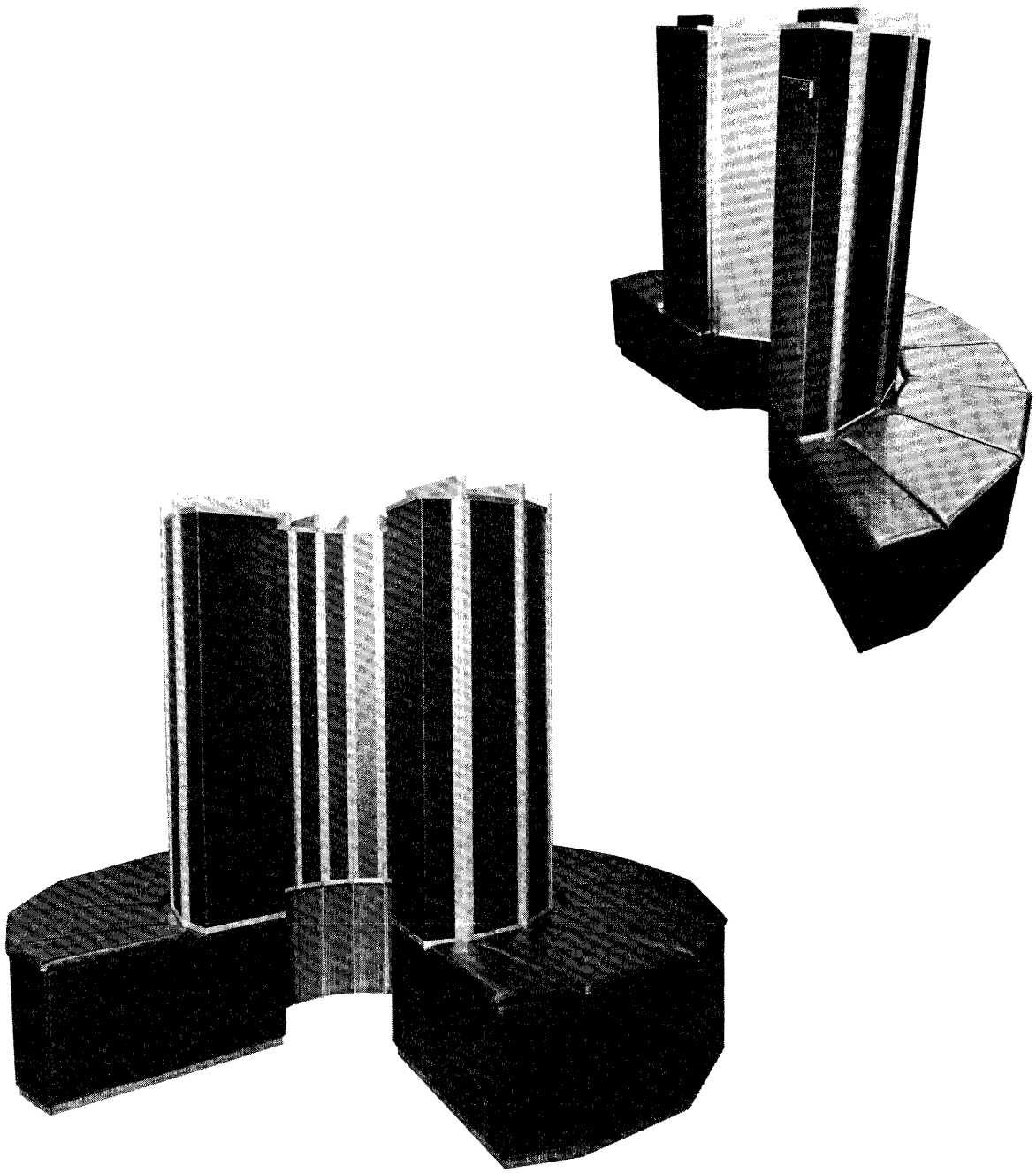


Figure 1-2. Central Processing Unit chassis variations

MAINTENANCE CONTROL UNIT

The Maintenance Control Unit (MCU) is used for supervisory operation and maintenance on CRAY-1 System Models S/250, S/500, and S/1000. It consists of a minicomputer, a tape drive, a card reader, a removable pack disk drive, a line printer, and 2 operator consoles. It may be used to enter jobs locally. Figure 1-3 shows the Maintenance Control Unit.

Figure 1-3. Maintenance Control Unit

INPUT/OUTPUT SUBSYSTEM

Models S/1200 through S/4400 of the CRAY-1 S Series are equipped with an I/O Subsystem composed of 2, 3, or 4 I/O Processors, a Buffer Memory, and required interfaces. Each I/O Processor (IOP) is independent and handles some portion of the I/O requirements for the system. I/O Processors are designed for fast data transfer between a front-end computer peripheral device and Buffer Memory or between Buffer Memory and the Central Processing Unit. Each I/O Processor has a computation section, a memory section, a control section, and an I/O section. Part 3 of this publication describes the I/O Processor in detail. One I/O Processor is assigned a supervisory role for the group of I/O Processors.

The computation section of an IOP has functional units and operand registers, and uses an accumulator in single-address operation.

The I/O Processor Memory holds 65,536 words of 16 bits each in bipolar integrated circuits; 16-bank phasing is used in this memory to maintain high-speed data transfers.

The I/O Processor section has six direct memory access (DMA) ports divided among the channels. A DMA port can transfer data at peak rates of approximately 850 Mbits per second; one port may be receiving data at the same time another port is sending data. Interfaces with internal buffering connect the I/O Processor to mass storage devices or to other high-speed equipment. Several independent channels may share one DMA port and this is done for some peripheral devices such as disk storage units. A standard set of peripherals provide for supervisory control and for maintenance: a tape drive, a line printer, and two operator consoles. Cray Research interfaces connect I/O Processors with front-end or host computers for network operation. Appendix F lists the various functions assigned to channels in a typical I/O Subsystem.

The Buffer Memory assists data transfer between peripheral devices and the Central Processing Unit. Buffer memory stores 1 million to 8 million words of 64 bits each, using single error correction/double error detection (SECDED) data protection. All I/O Processors share the Buffer Memory. The Central Processing Unit is not directly connected to the Buffer Memory. An I/O Processor relays data between the Buffer Memory and the CPU.

The I/O Subsystem is housed in a 4-column chassis similar to the CPU chassis. Figure 1-4 shows this chassis.

MASS STORAGE

The basic unit of mass storage for the CRAY-1 Series Computer Systems is the DD-29 Disk Storage Unit (DSU). This is a 600 Mbyte disk drive having a maximum data transfer rate of 35.4 Mbits per second.

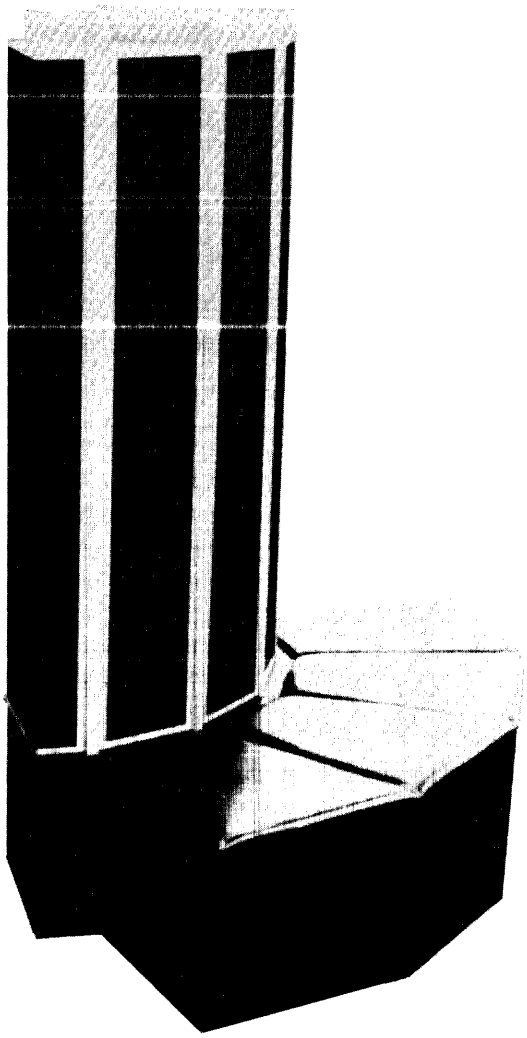


Figure 1-4. I/O Subsystem

On Models S/250, S/500, and S/1000, up to four DD-29s can be connected to one DCU-3 Disk Controller Unit (figure 1-5). A mass storage subsystem composed of 2 to 8 DCU-3 Disk Controller Units and 2 to 32 Disk Storage Units can be configured on the Models S/250, S/500, and S/1000. However, the actual number of units depends on the number of available I/O channel pairs (e.g., if 4 channel pairs are used from front-end computer systems, including 1 for the MCU, the remaining 8 can be used for mass storage.) Operating and programming information for the DCU-3 is included in CRI Publication HR-0630.

On models S/1200 through S/4400, up to four DD-29s can be connected to one DCU-4 Disk Controller Unit. The DCU-4 Controller Unit interfaces the four disk units with the I/O Processor through two DMA ports. The I/O Processor and the Disk Controller Unit are fast enough to keep all four DSUs operating at full speed without missing data or skipping revolutions. A minimum of 2 and a maximum of 48 DD-29s can be configured. Figure 1-6 shows a DD-29 Disk Storage Unit. The DCU-4 Disk Controller Unit is housed in the I/O Subsystem chassis.

Each DD-29 Disk Storage Unit has two ports for controllers. A second independent data path to each disk storage unit may exist through another Cray Research controller. Reservation logic is provided to control access to each disk storage unit.

Operational characteristics of the DD-29 Disk Storage Units are summarized in table 1-3. Further information about the DCU-4 and DD-29 is presented in part 3, section 7 of this publication.

Table 1-3. Characteristics of a DD-29 Disk Storage Unit

Bit capacity per drive	4.848×10^9	Maximum Latency	16.6 ms
Tracks per surface	823	Access time	15 - 80 ms
Sectors per track	18	Data transfer rate	38.7×10^6 bits/s
Bits per sector	32,768		
CPU words per sector	512		
Head groups per drive	10	Bits per cylinder	5.9×10^6
Recording surfaces per drive	40	CPU words per cylinder	92,160

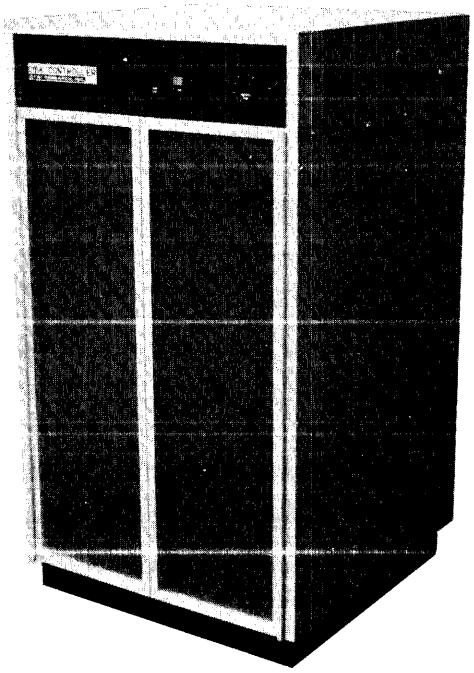


Figure 1-5. DCU-3 Disk Controller Unit

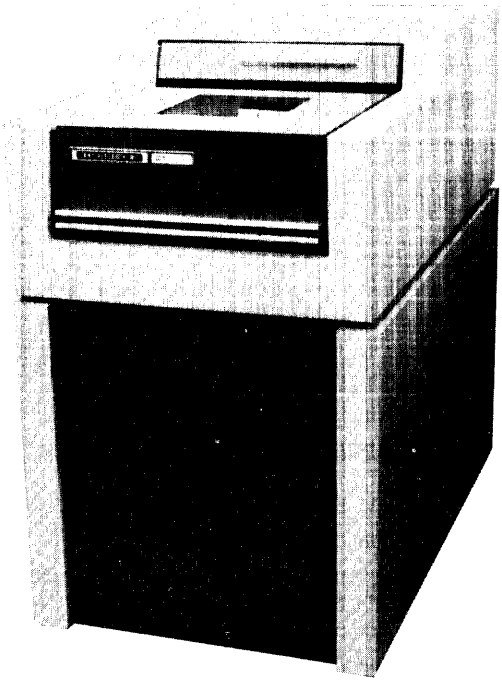


Figure 1-6. DD-29 Disk Storage Unit

CONDENSING UNITS

The condensing units contain the major components of the refrigeration system used to cool the computer chassis. Heat is removed from the condensing unit by a secondary cooling system that is not part of the CRAY-1 Computer System. Figure 1-7 shows the condensing unit.

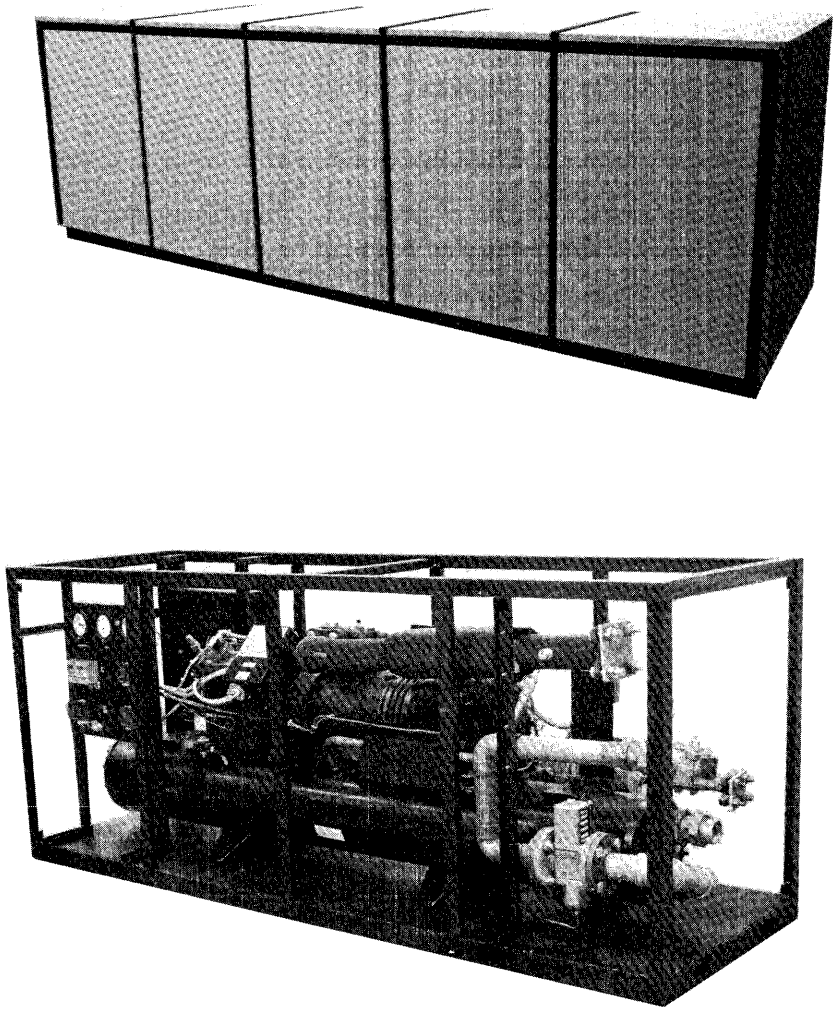


Figure 1-7. Condensing Unit

POWER DISTRIBUTION UNITS

The Central Processing Unit, I/O Subsystem, and disk controller units all operate from 400 Hz 3-phase power. The Power Distribution Unit (PDU) for the CPU contains adjustable transformers to regulate the voltage to each power supply for the CPU. The PDU also contains temperature monitoring equipment that checks temperatures at strategic locations on the CPU chassis. Automatic warning and shutdown circuitry protect the mainframe in case of overheating or excessive cooling. The control switches for the motor-generators and the Condensing Unit are mounted on the CPU Power Distribution Unit.

The Power Distribution Unit for the I/O Subsystem performs similar functions for the I/O Subsystem chassis.

The disk controller units have these functions built into the disk controller unit cabinet.

Figure 1-8 shows the Power Distribution Units for the CPU and for the I/O Subsystem.

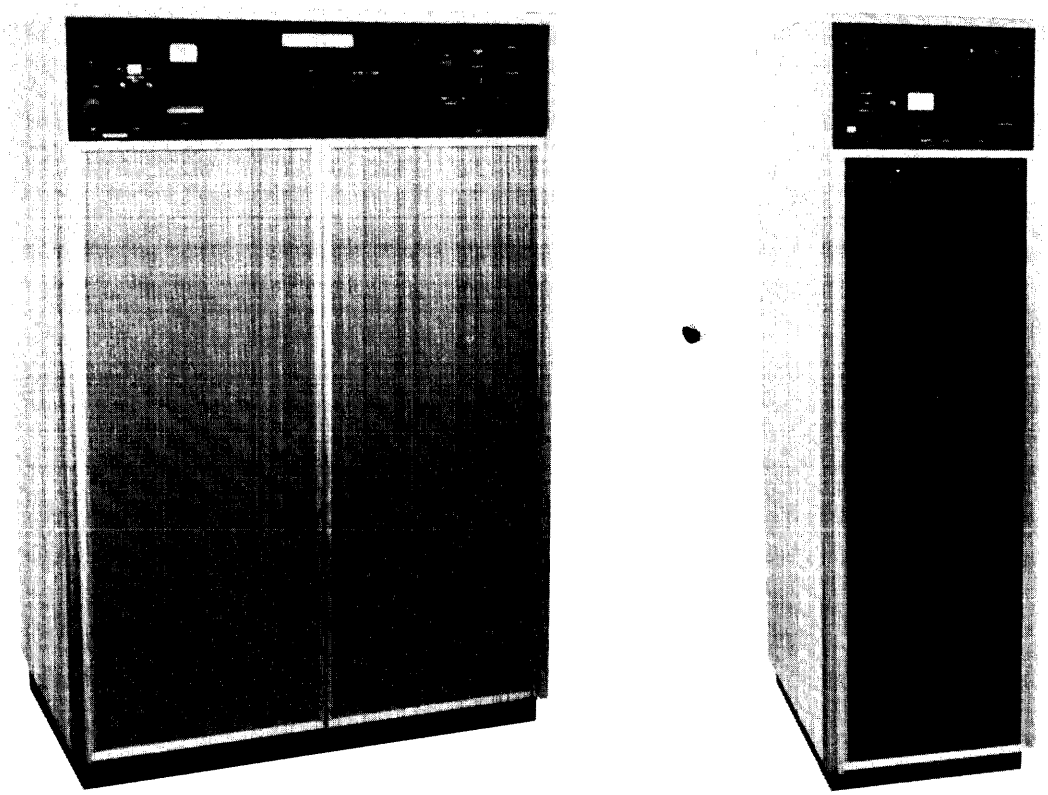


Figure 1-8. Power Distribution Units

MOTOR-GENERATORS

The Motor-generators (MGs) units convert primary power from the commercial mains to the 400 Hz power used by the computer system. They isolate the system from transients and fluctuations on the commercial power mains. The equipment consists of two or three MGs and a control cabinet. Figure 1-9 shows a Motor-generator and the control cabinet.

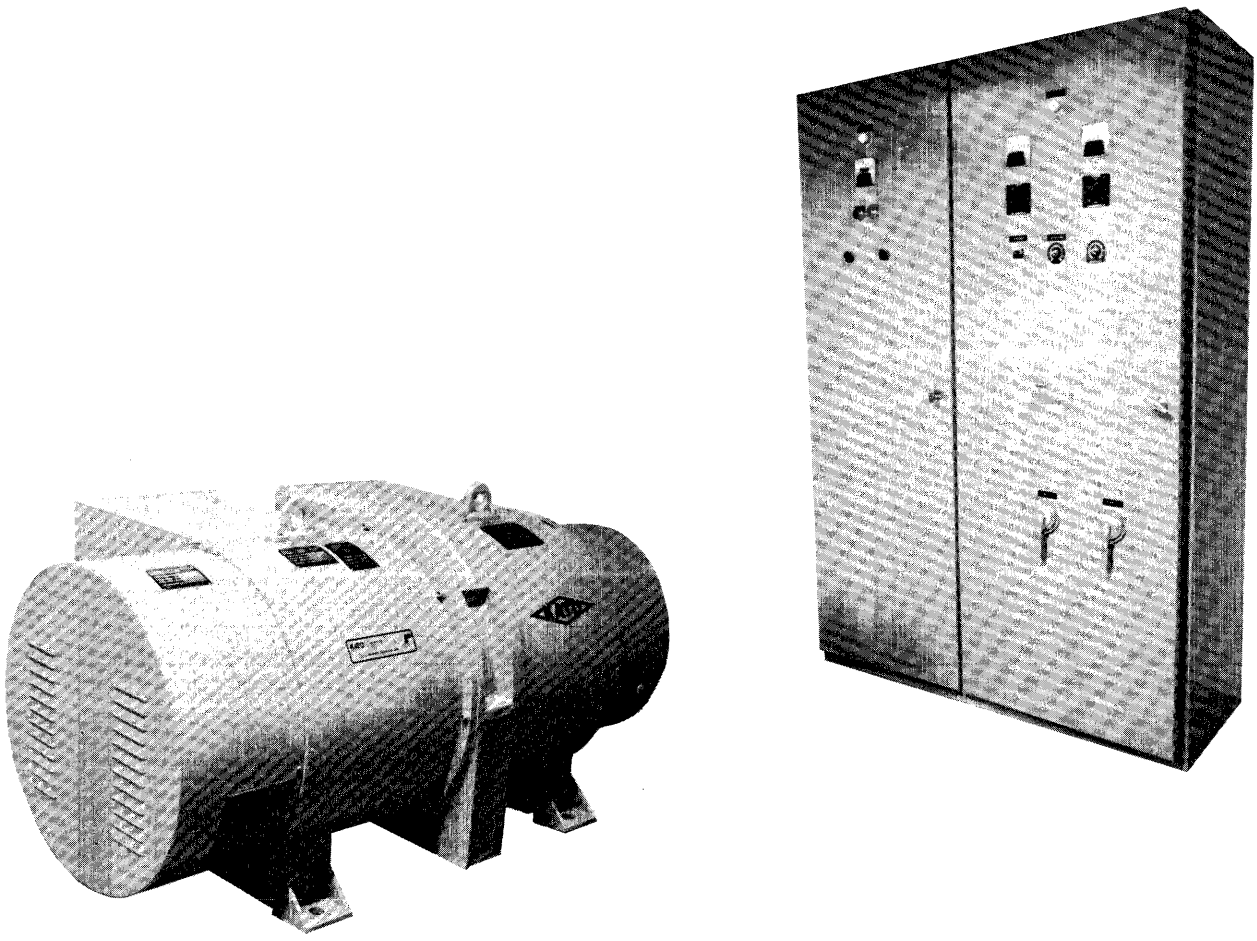


Figure 1-9. Motor-generator equipment

SYSTEM CONFIGURATION 2

INTRODUCTION

Several combinations of the basic system components are supported in the S Series of CRAY-1 Computer Systems. The Central Memory of the Central Processing Unit (CPU) is available in several different sizes. The I/O Subsystem is present on larger models in the S Series and consists of two to four processors. The following paragraphs describe the models available in the CRAY-1 S Series.

S/250, S/500, AND S/1000 MODELS

The S/500 has a 1/2-million-word Central Memory, and the S/1000 a 1-million word Central Memory. The S/250, which is a discontinued model, has a 1/4-million-word Central Memory. Figure 2-1 shows these types of configurations.

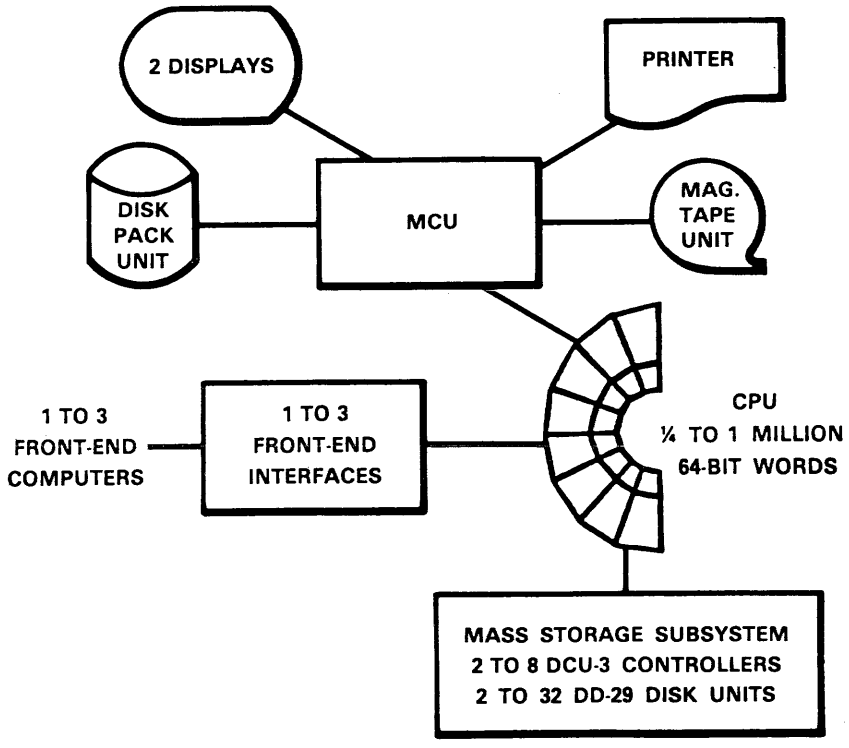


Figure 2-1. Block diagram of S/250, S/500, and S/1000 systems

All of these systems use the Maintenance Control Unit for supervision and maintenance. This consists of a minicomputer supporting the standard group of peripherals: a magnetic tape unit, a line printer, a removable pack disk drive, and two operator consoles. The mainframe of the S/250, S/500 and S/1000 models have eight chassis columns as standard. Up to three front-end computer interfaces and up to 32 disk storage units can be connected to the Central Processing Unit via the CRAY-1 I/O channels.

S/1200, S/2200, S/4200 MODELS

The S/x200 systems all have the common characteristic of a 2-processor I/O Subsystem. They differ in size of the Central Memory: S/1200 has 1 million words; S/2200 has 2 million words; and S/4200 has 4 million words. The S/1200 and S/2200 CPU chassis has 8 columns as standard. The S/4200 CPU chassis has 12 columns as standard. Figure 2-2 shows a configuration for these systems.

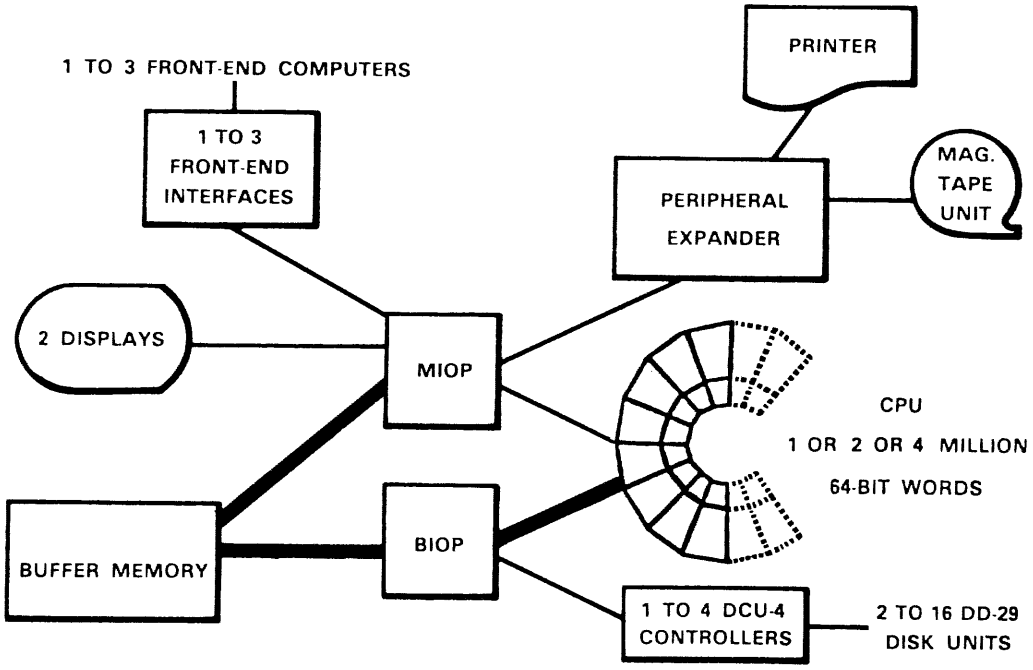


Figure 2-2. Block diagram of S/1200, S/2200, and S/4200 systems

The Master I/O Processor (MIOP) controls the front-end interfaces and the standard group of station peripherals. The Peripheral Expander interfaces the station peripherals to one DMA port of the MIOP. The Master I/O Processor also connects to the Buffer Memory and to the CPU over a CRAY-1 channel pair. Data from the front-end computer goes to Buffer Memory via the MIOP, and then to the CPU via the Buffer I/O Processor. The Master I/O Processor communicates with the CPU operating system to coordinate the activities of the I/O Subsystem.

The Buffer I/O Processor (BIOP) is the main link between the Central Processing Unit and the mass storage devices. Data from mass storage is often transferred through the BIOP I/O Memory to the Buffer Memory. Then the data is returned to the BIOP I/O Memory and sent to the CPU. The BIOP is the only I/O Processor having a standard Memory Channel to the CPU. The Memory Channel operates at speeds of approximately 800 Mbits per second. A second Memory Channel to another IOP is optional. A Model S/x200 supports up to 16 disk storage units.

S/1300, S/2300, S/4300 MODELS

The S/x300 systems all have the common characteristic of a 3-processor I/O Subsystem. They differ in size of the Central Memory: S/1300 has 1 million words; S/2300 has 2 million words; and the S/4300 has 4 million words. The S/1300 and S/2300 CPU chassis has 8 columns as standard. The S/4300 CPU chassis has 12 columns as standard. These configurations are the same as those described previously, except for the addition of a third I/O Processor. The third I/O Processor can be used either to control block multiplexer channels or to handle additional disk storage units. These two configurations are shown in figures 2-3 and 2-4, respectively.

The Auxiliary I/O Processor (XIOP) used for block multiplexer channels interfaces to a maximum of four BMC-4 Block Multiplexer Controllers, each of which can handle up to four block multiplexer channels. The XIOP uses one DMA port for each controller and another DMA port to connect with the Buffer Memory. Thus, data flow from the block multiplexer channel goes through the XIOP to the Buffer Memory, then from the Buffer Memory through the Buffer I/O Processor to the CPU.

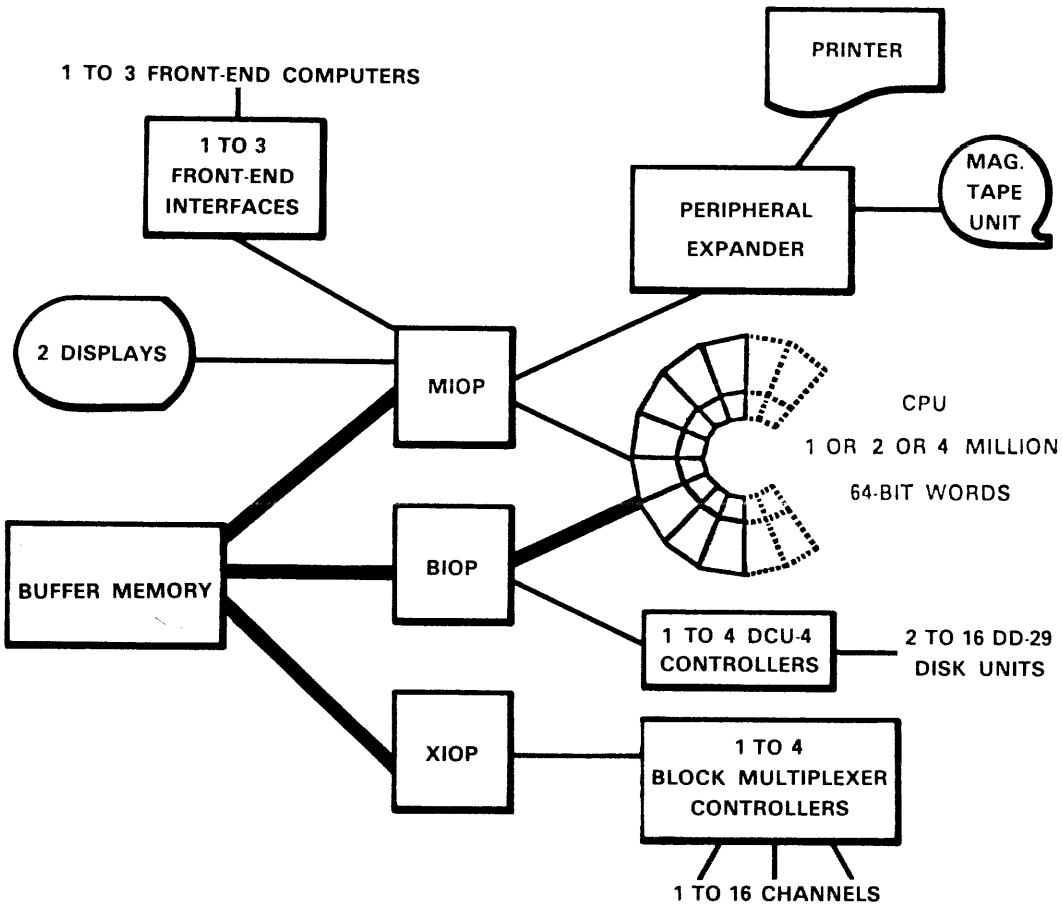


Figure 2-3. Block diagram of S/1300, S/2300, and S/4300 systems with block multiplexer channels

When the third I/O Processor is used for additional disk storage units, it is called a Disk I/O Processor, or DIOP. This processor can handle up to four disk controller units with up to 16 disk storage units. This addition effectively doubles the mass storage capacity over that of the S/X200 models--up to 32 disk storage units are possible. The disk data is transferred through the DIOP into Buffer Memory and then to the Central Processor through the Buffer I/O Processor.

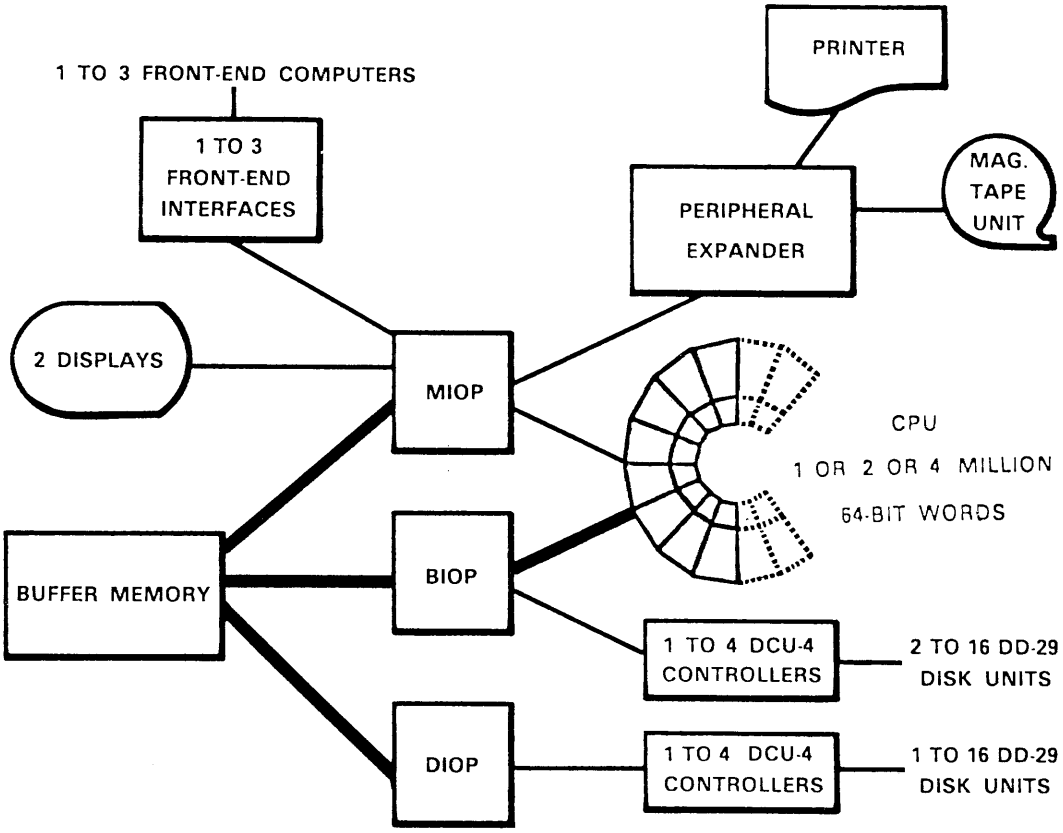


Figure 2-4. Block diagram of S/1300, S/2300, and S/4300 systems with increased disk capacity

S/1400, S/2400, S/4400 MODELS

The S/x400 systems all have the common characteristic of a 4-processor I/O Subsystem. They differ in the size of Central Memory: the S/1400 has 1 million words; the S/2400 has 2 million words; and the S/4400 has 4 million words. The S/1400 and S/2400 CPU chassis has 8 columns as standard. The S/4400 CPU chassis has 12 columns as standard. Figures 2-5 and 2-6 show these configurations. In these configurations, the third I/O Processor is assigned to Block Multiplexer Controllers or disk storage units and the fourth processor handles additional disk storage units. A MIOP and a BIOP are used as in the configurations described previously. These two configurations are shown in figure 2-5 for the block multiplexer channels, and figure 2-6 for the increased disk capacity. Each configuration is a permanent, factory-wired configuration.

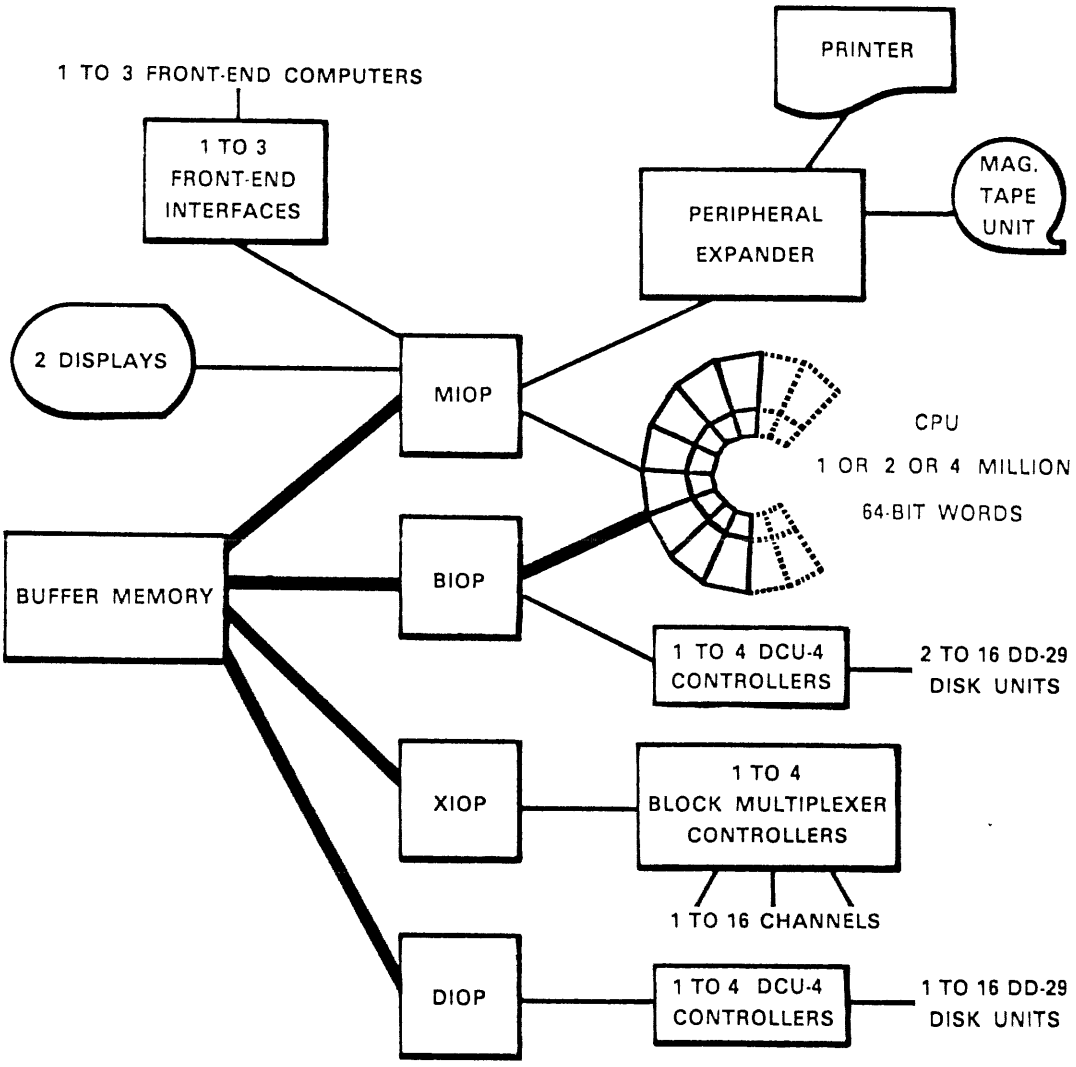


Figure 2-5. Block diagram of S/1400, S/2400, and S/4400 systems with block multiplexer channels

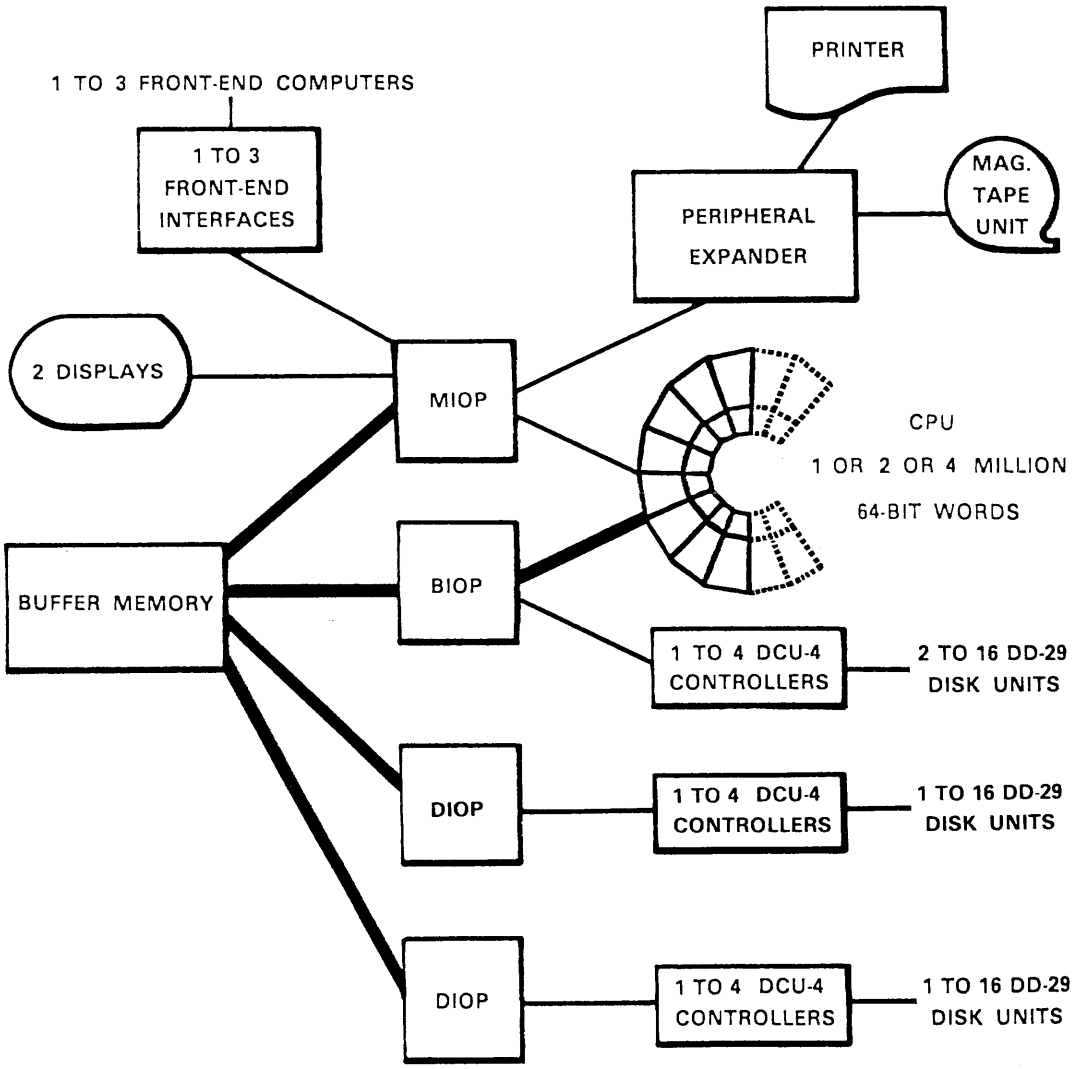


Figure 2-6. Block diagram of S/1400, S/2400, and S/4400 systems with increased disk capacity

When the third I/O Processor is used as an auxiliary I/O Processor (XIOP) for block multiplexer channels, it can handle up to 16 channels via a maximum of four Block Multiplexer Controllers. Channel data flows to the CPU via the Buffer Memory and the Buffer I/O Processor.

When the third I/O Processor is used for expanded disk capacity, it is called a Disk I/O Processor, DIOP. This configuration makes available the maximum mass storage resource. Up to 48 disk storage units can be controlled by a BIOP and two DIOPs.

MAINTENANCE CONTROL UNIT

The S/250, S/500, and S/1000 CRAY-1 Computer Systems are each 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 Maintenance Control Unit (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 includes:

1. A 16-bit minicomputer with 32K words of 16-bit memory
2. A 132-column line printer
3. An 800 bpi 9-track tape unit
4. Two display terminals
5. A removable pack disk drive

Included with the MCU system is a software package that enables it to serve as a local batch station during production hours. As a local station, it may be used to submit diagnostic routines for execution or to submit 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 deadstart sequence. This sequence is described in detail with the CPU control section.

Detailed information about the MCU is presented in separate publications.

INTERFACES TO FRONT-END COMPUTER

A front-end computer system is a self-contained system that executes under the control of its own operating system. The CRAY-1 computer systems is interfaced to 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 peripheral equipment. The interfaces compensate for differences in channel widths, machine word size, electrical logic levels, and control protocols. The MIOP communicates through a CRAY-1 I/O channel pair to a channel adapter module in the CPU chassis. The channel adapter module drives long cables of up to 300 feet. The interface cabinet is usually placed near the front-end computer; the CRAY-1 and the front-end computer cables connect to the interface cabinet. The standard interfaces are described in part 2, section 7.

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.

Peripheral equipment attached to the front-end computer varies depending on the use of the System.

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 not presented in this publication.

SYSTEM OPERATION

The overall system consists of the components as described previously, the communication paths among them, and the software that moves the data within the devices. The following paragraphs briefly describe the system communication and job flow for systems that include an I/O Subsystem. Following that is a description of the deadstart process used to bring the system to an operational state.

I/O SUBSYSTEM COMMUNICATION

The CRAY-1 S Series system provides communication paths between the Central Processing Unit and two I/O Processors, between each I/O Processor and the Buffer Memory, and among all the I/O Processors. The arrangement is shown in figure 2-7.

Communication between the Central Processing Unit and the I/O Processors is over one approximately 50 Mbits per second CRAY-1 I/O channel pair to the Master I/O Processor, and over one approximately 850 Mbits per second Memory Channel to the Buffer I/O Processor. The CRAY-1 I/O channel pair is used for exchanging system control information with the Master IOP, while the Memory Channel transfers data through the Buffer IOP.

One DMA port of each I/O Processor is connected with the Buffer Memory through an approximately 800 Mbit/second channel. The Buffer Memory is used to organize data from one I/O Processor and store it until the Buffer I/O Processor can remove that data and pass it to the Central Processing Unit. In this way, each I/O Processor communicates with every other I/O Processor in high-speed data block transfers.

Additionally, each I/O Processor is connected with the other I/O Processors by slower channels called "accumulator channels". These channels pass one 16-bit parcel at a time from the accumulator of one I/O Processor to the accumulator of another I/O Processor. These are used primarily for control and status reporting.

Any errors occurring in the system memories or Memory Channel are reported to the Master I/O Processor via special error channels that are separate from the data channels. Thus all error handling for the system is initiated by the Master I/O Processor.

The resulting communications network among the processors speeds the flow of data from the front-end computers, peripheral devices and mass storage units; it stores the data as necessary; and it passes the data to the CPU. The network also facilitates transfer of results from the CPU to the final destination.

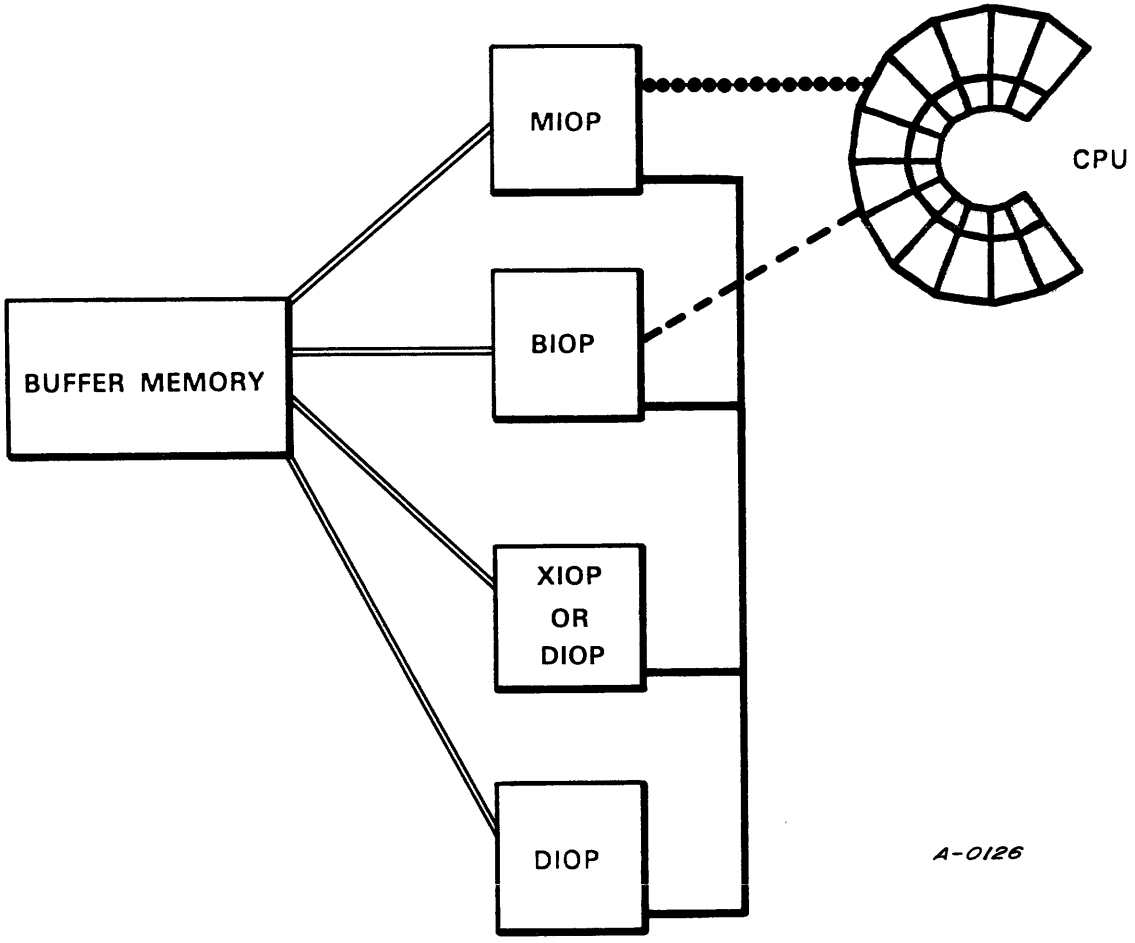


Figure 2-7. I/O Subsystem communication

JOB FLOW

Figure 2-8 shows a simplified view of the job flow within the system; numbers index the sequence of operations. Jobs originate in the front-end computer network, with the front-end computer passing (1) job control statements, programs, and data files to the Master I/O Processor. The Master I/O Processor transfers the job (2) to the Buffer Memory. The MIOP informs the CRAY-1 Operating System (3) of the existence of data in the Buffer Memory. The CRAY-1 Operating System stores data by a request (4) to the MIOP to have the DIOP store the data. The MIOP commands the Disk I/O Processor (5) to transfer data from the Buffer Memory (6) to mass storage (7).

The operating system analyzes the resources required for the job, and when these are available, calls the Master I/O Processor for the job (8). The MIOP tells the DIOP (9) to bring the job in from mass storage (10) and store it in Buffer Memory (11) for transfer to the Buffer IOP. As soon as possible, the Buffer IOP begins transferring the job from Buffer Memory (12) to its own I/O Memory and then to the Central Processing Unit (13) over the Memory Channel. When the transfer is complete, the CPU begins executing the job control statements.

Several transfers to and from mass storage may be necessary during job execution. Output from the job is sent to mass storage (14 - 20), and when the job is finished, the MIOP is notified. The Master I/O Processor tells the Disk I/O Processor (21) to bring in the results from mass storage (22) and place them in Buffer Memory (23). The operating system tells the MIOP (24) to take data from the Buffer Memory (25) and pass it to the front-end computer (26).

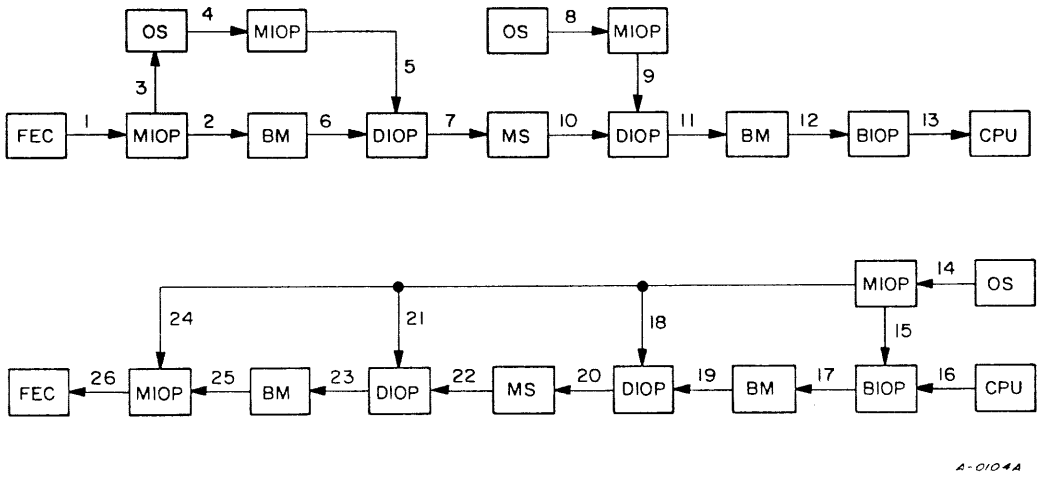


Figure 2-8. Job flow diagram

DEADSTART

A model S/1200 through S/4400 system is initially started from a magnetic tape bootstrap which loads into the memory of the Master I/O Processor. This program is enough to load further software from the tape and store it in Buffer Memory. When the Buffer Memory has the deadstart program, the Master I/O Processor commands the Buffer I/O Processor to deadstart from Buffer Memory. The Buffer I/O Processor loads the program which enables it to load further program data from a disk storage unit. The disk data is passed through the Buffer Memory to the Master I/O Processor which can completely initiate the I/O Subsystem. Then, under control of the Master I/O Processor, the main portion of the operating system is loaded into the Central Memory and the system becomes operational.

Initial installation of the deadstart program and operating system on the disk storage unit is done by maintenance personnel from magnetic tape. In the case of a failure in the Master I/O Processor, a maintenance deadstart panel can be used to load a deadstart or diagnostic program into the Master or Buffer I/O Processor.

For a Model S/250, S/500, or S/1000 system using a Maintenance Control Unit, deadstarting occurs from the removable-pack disk storage unit through the minicomputer. The main operating system is then loaded from the same disk storage unit into Central Memory. This operation is described with the control section of the Central Processing Unit.

PART 2

CENTRAL PROCESSING UNIT

GENERAL INFORMATION 1

INTRODUCTION

The Central Processing Unit (CPU) is the computer that executes programs, runs user jobs, and oversees the job flow within the CRAY-1 S Series Computer System. Scalar and vector processing capabilities are combined with large, fast Central Memory and high-volume I/O channels. Figure 1-1 represents the basic organization of the Central Processing Unit. The memory is expandable from one-quarter million to four million words of 64-bits each. The Memory Channels are used for high-speed data transfers to and from I/O Processors in the I/O Subsystem. Twelve I/O channel pairs provide access to front-end computers, mass storage controllers, and to the I/O Subsystem.

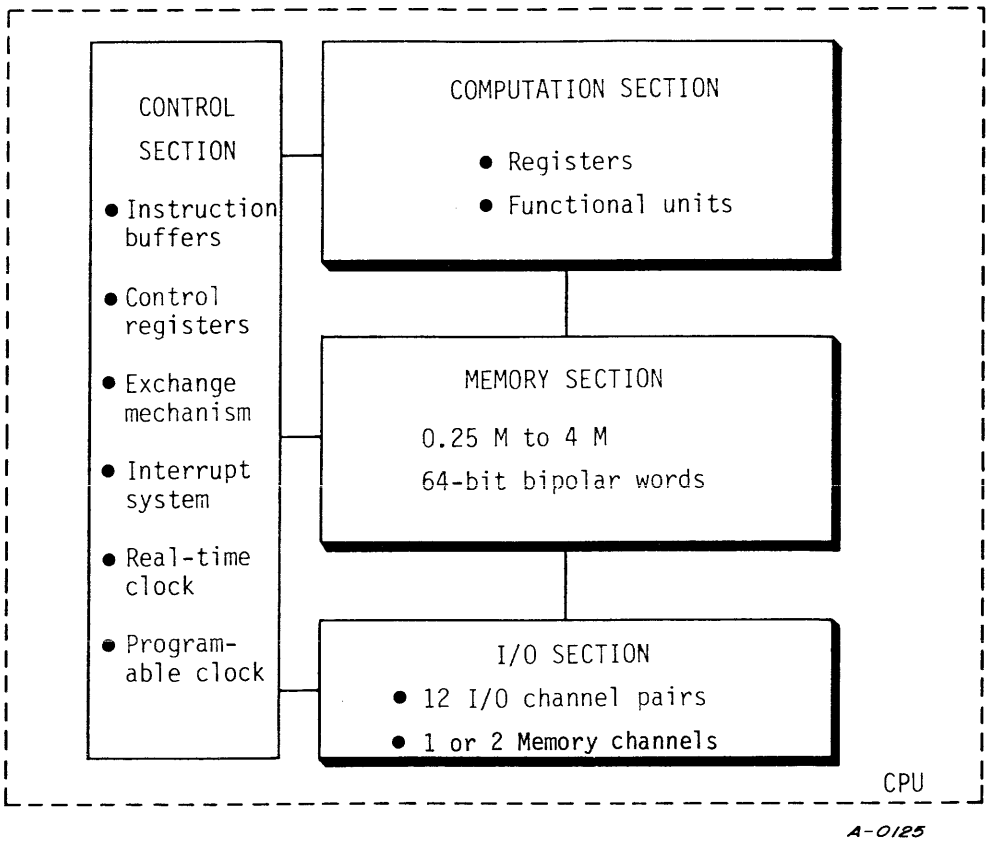


Figure 1-1. Basic organization of the CPU

The following paragraphs provide additional general information about the four sections of the CPU; later sections describe the features in greater detail.

Figure 1-2 illustrates the components of the CPU and presents a generalized view of the flow of data in the system.

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 right to left as powers of 2, starting with bit 2^0. Bit 2^{63} of an S, V, or T register value represents the most significant bit in the operand. Bit 2^{23} of an A or B register value represents the most significant bit in the operand.

NUMBER CONVENTIONS

Unless otherwise indicated, numbers in this manual are decimal numbers. Octal numbers are indicated with an 8 subscript. Exceptions are register numbers, channel numbers, and instruction forms, which are given in octal without the subscript.

CLOCK PERIOD

The basic unit of CPU computation time is 12.5 nanoseconds and is referred to as a clock period (CP). Instruction issue, memory references, and other timing considerations are often measured in clock periods.

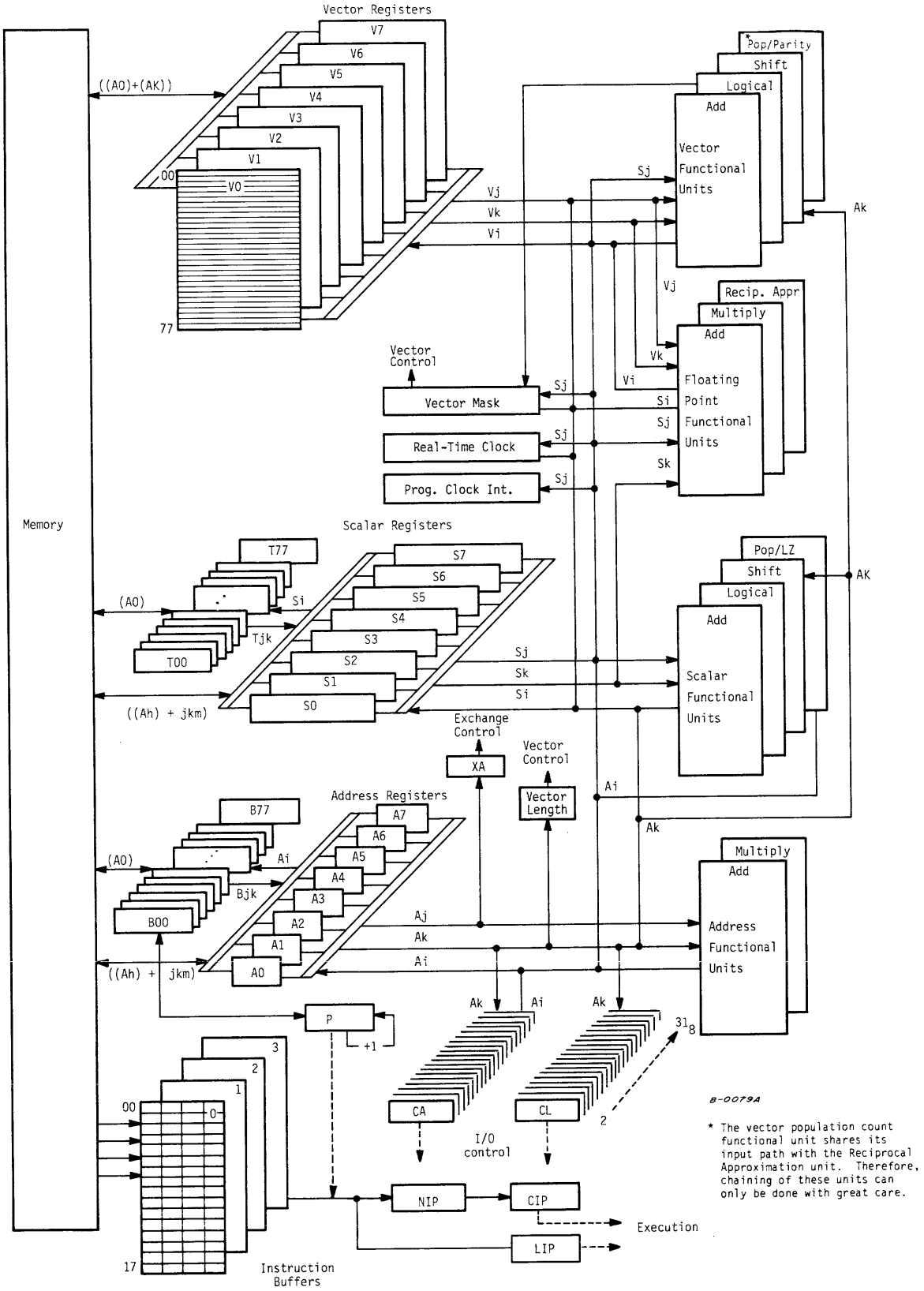


Figure 1-2. Control and data paths in the CPU

MEMORY SECTION

The memory for the CRAY-1 consists of 8 or 16 banks of bipolar LSI memory. Four memory size options are available: 524,288 or 1,048,576 or 2,097,152 or 4,194,304 words. A 226,144-word memory is no longer available. Each word is 72 bits--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 4 CPs (50 ns). The access time, that is, the time required to fetch an operand from memory to an operational register, is 11 CPs (137.5 ns). There is no inherent memory speed degradation for 16-bank memories of less than 4 million words.

The maximum transfer rate for B, T, and V registers is one word per CP. For A and S registers, it is one word per 2 CPs. Transfer of instructions to the instruction buffers occurs at a rate of 16 parcels (four words) per CP.

Thus, the CPU memory sizes support the requirements of scientific applications, while the 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.

Table 1-1 summarizes the features of the CPU memory section. CPU Memory is described in detail in section 2.

Table 1-1. Characteristics of the CPU memory section

- From 0.25M to 4M words of bipolar integrated circuit memory
- 64 data bits and 8 error correction bits per word
- 8 or 16 interleaved banks
- 4 CP bank cycle time
- 1 word per CP transfer rate to B, T, and V registers
- 1 word per 2 CP transfer rate to A and S registers
- 4 words per CP transfer rate to instruction buffers
- Single error correction/double error detection (SECDED)

CONTROL SECTION

The control section performs all decisions related to instruction issue and coordinates the activities for the three types of processing: address, scalar, and vector.

The control section executes 128 basic instruction codes as either 16-bit (1 parcel) or 32-bit (2 parcel) instructions and provides for register reservation, memory field protection, memory access, and interrupt control.

Table 1-2 summarizes features of the control section. This section is described in greater detail in section 3.

Table 1-2. Characteristics of the CPU control section

- 12.5 nanosecond clock period (CP)
- 4 instruction buffers of 64 16-bit parcels each
- 128 basic instruction codes
- Program exchange mechanism
- Error/monitor interrupt flags
- Memory and program field protection

COMPUTATION SECTION

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

Eight address (A) registers are used to store 24-bit integers or addresses. Sixty-four intermediate address (B) registers store data for use by the A registers. Eight scalar (S) registers store 64-bit operands for scalar operations. Sixty-four intermediate (T) registers store data temporarily for the S registers. Eight vector (V) registers are made up of 64 elements in each register. Each element stores one 64-bit operand. The vector registers are used in vector processing.

A series of operands can be an ordered set, called a vector. A vector instruction operates on a series of operands, doing the same function repetitively, and producing a series of results. Scalar processing starts an instruction, handles one operand or operand pair, then stops the operation. The main advantage of vector over scalar processing is the elimination of the instruction start-up time for all but the first operand.

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

Floating-point instructions provide for addition, subtraction, multiplication, and reciprocal approximation. The reciprocal approximation instructions allow a floating-point divide operation using a multiple instruction sequence. These instructions produce 64-bit results.

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; the operation is accomplished through a software algorithm using floating-point hardware.

The instruction set includes Boolean operations for OR, AND, equivalence, 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, most 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.

Population and parity counts are provided for both vector and scalar operations. Additionally, scalar operations may include leading zero counts.

Table 1-3 summarizes the characteristics of the CPU computation section.

Table 1-3. Characteristics of CPU computation section

- Integer and floating-point arithmetic
- Twos complement integer arithmetic
- Sign and magnitude floating-point arithmetic
- Address, scalar, and vector processing modes
- Thirteen 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

INPUT/OUTPUT SECTION

One or two Memory Channels transfer data between Central Memory and the Buffer I/O Processor. Each channel is 64 bits wide and uses 8 check bits with each word. Data words are transferred in blocks of 16 under control of Data Ready and Data Transmit control signals. A maximum transfer rate of approximately 850 Mbits per second is possible on this channel.

Normal input and output communication with the CPU is over 12 full duplex 16-bit channel pairs. Associated with each channel are control lines that indicate the presence of data on the channel.

Table 1-4 summarizes features of the channels in the CPU input/output section. Channels are described in detail in section 5.

Table 1-4. Characteristics of the CPU input/output section

Up to twelve I/O channel pairs; 50 or 160 Mbits/s maximum rate
- Four channel groups containing either 6 input or 6 output channels
- Channel groups served equally by memory (scans each group every 4 CPs)
- Channel priority resolved within channel groups
- 16 data bits, 3 control bits, and 4 parity bits in each direction
- Lost data detection
One or two Memory Channels; approx. 800 Mbits/s maximum rate each
- 64 data bits, 3 control bits, and 8 check bits in each direction

CENTRAL MEMORY SECTION 2

INTRODUCTION

The Central Memory of the CPU consists of 8 or 16 banks of bipolar LSI memory. Four memory sizes are available:

524,288 words - 8 banks,
1,048,576 words - 8 banks,
2,097,152 words - 8 banks,
4,194,304 words - 16 banks.

The banks are independent of each other. A 262,144-word 8-bank memory is no longer available.

MEMORY CYCLE TIME

The memory cycle time is 4 CPs (50 ns). The access time, that is, the time required to fetch an operand from memory to an operational register, is 11 CPs (137.5 ns).

MEMORY ACCESS

The Central Memory of the CPU 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.

Block transfers require all memory requests to be completed before the block transfers can issue. Once issued, they inhibit 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 slowing or disallowing the vector operations that would be affected by the slowed memory referencing rate. The vector logic identifies 1/4 speed (4 CPs), 1/2 speed (2 CPs), and full speed (1 CP) data rates from memory.

Fetch operations bring instructions from Central Memory to the instruction buffers. Fetch operations require completion of all other types of memory references before the fetch operations reference memory. Once the fetch request is honored, all other types of memory reference are inhibited.

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

Scalar and I/O memory references are examined in three registers for possible memory conflicts. These three registers contain the low-order 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 register is rank A, the second is rank B, and the third is rank C. At each clock period, 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 8 CPs 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 in CP 2 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 or request
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 in memory in the same clock period.

A scalar conflict in rank A (CP 2 of a scalar instruction) causes a hold storage on this instruction for 3 CPs. 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 2 CPs to issue a scalar reference instruction.

A scalar conflict in rank B (CP 3) causes a hold storage on this instruction for 2 CPs. 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 1 CP. There is also a hold issue signal, which blocks issue of another scalar reference instruction.

The Memory Channel shares the same access with the normal I/O channels, but the normal I/O channels have priority. The Memory Channel operates in blocks of 16 words with a 1-CP pause between blocks to allow other memory operations to break the Memory Channel transfer.

Under normal operating conditions on codes performing a mix of vector and scalar instructions, the memory access supports four disk and three interface channel pairs 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 degradation is caused by the delays imposed on the issue of vector memory instructions because block transfers require memory quiet before issue.

MEMORY ORGANIZATION

Central Memory is organized into 8 or 16 banks to minimize memory conflicts and to exploit the speed of the memory chips. In a 16-bank machine, each bank occupies half a column 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 8-bank organization is standard on the 8-column mainframes. The 16-bank phasing is standard on the S/4200 and larger mainframes. A maintenance feature for all 16-bank systems permits them to operate with only 8 banks using either the left or right half of memory. This is accomplished by installing two special modules and setting the bank select switch (on the Power Distribution Unit) to the left or right banks.

MEMORY ADDRESSING

A word in a 16-bank memory is addressed in a maximum of 22 bits as shown in figure 2-1. The low-order 4 bits specify one of the 16 banks. The next field specifies an address within the chip. The high-order bits specify one of the chips on the module.

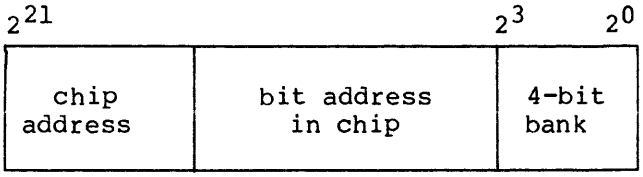


Figure 2-1. Memory address (16 banks)

A word in an 8-bank memory is addressed in a maximum of 21 bits as shown in figure 2-2. In this case, the low-order 3 bits specify one of the eight banks. The next field specifies an address within the chip. The high-order bits specify one of the chips on the module.

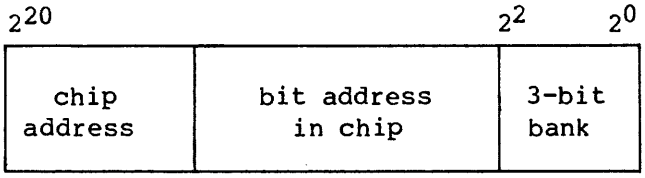


Figure 2-2. Memory address (8 banks)

SPEED CONTROL

For vector read and vector store instructions, the low-order 4 bits of (Ak) determines speed control (table 2-1).

Table 2-1. Vector memory rate x 80 x 10^6 references per second

Phasing	Increment or multiple in (Ak)							
	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 by eight 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

A 16-bank system can be readily modified to run on either the right or left 8 banks for maintenance purposes. This is accomplished by replacing two modules and setting the bank select switch on the Power Distribution Unit to the right or left banks. If the situation warrants it, the machine can continue running on one half of memory while repairs are made to the other half.

The effect of 8-bank phasing on instruction fetches is a predictable increase of 4 CPs 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 other differences, refer to the preceding paragraphs on Memory Addressing and Speed Control.

MEMORY 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 (figure 2-3).

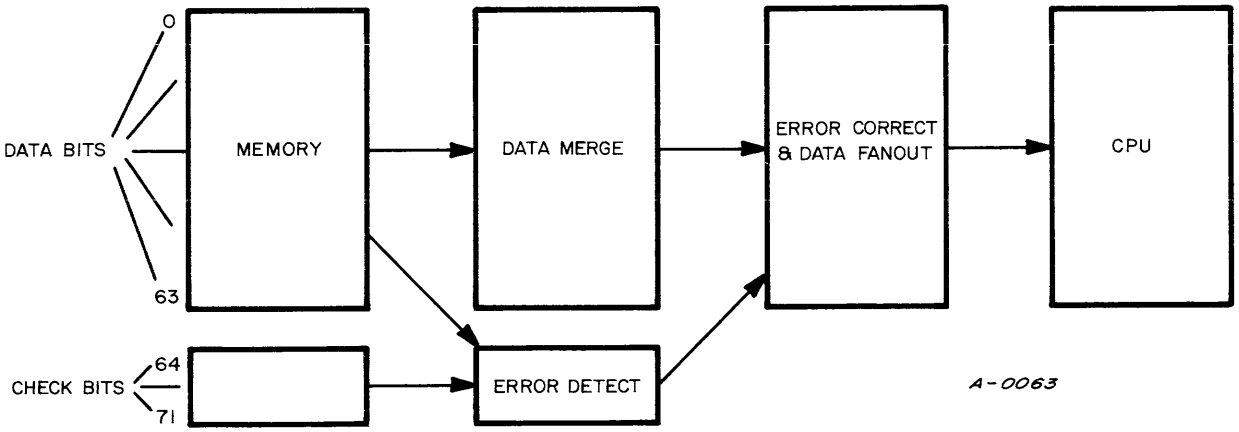


Figure 2-3. Memory data path with SECDED

The network operates on the basis of single error correction, double error detection (SECDED). If 1 bit of a data word is altered, the single error alteration is automatically corrected before passing the data word to the computer. If 2 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 3 or more bits in error, results are ambiguous.

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 8 check bits are each generated as even parity bits for a specific group of data bits. Figure 2-4 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 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 8 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 2-4 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 8 parity bits are called syndrome[§§] bits (S bits), shown as bits 64 through 71 in figure 2-4. The states of these S bits are all symptoms of any error that occurred. If all syndrome bits are 0, no memory error occurred.

Any change of state of 1 data bit will cause an odd number of syndrome bits to be set to 1. 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 five rules:

1. If all syndrome bits are 0, no error occurred.
2. If only 1 syndrome bit is 1, the associated check bit is in error.

---

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

§§ Syndrome: Any set of characteristics regarded as identifying a certain type, condition, etc. Websters New World Dictionary.

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

4. If more than 1 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. If 3 or more memory bits are in error, the parity of all syndrome bits is odd and results are ambiguous.

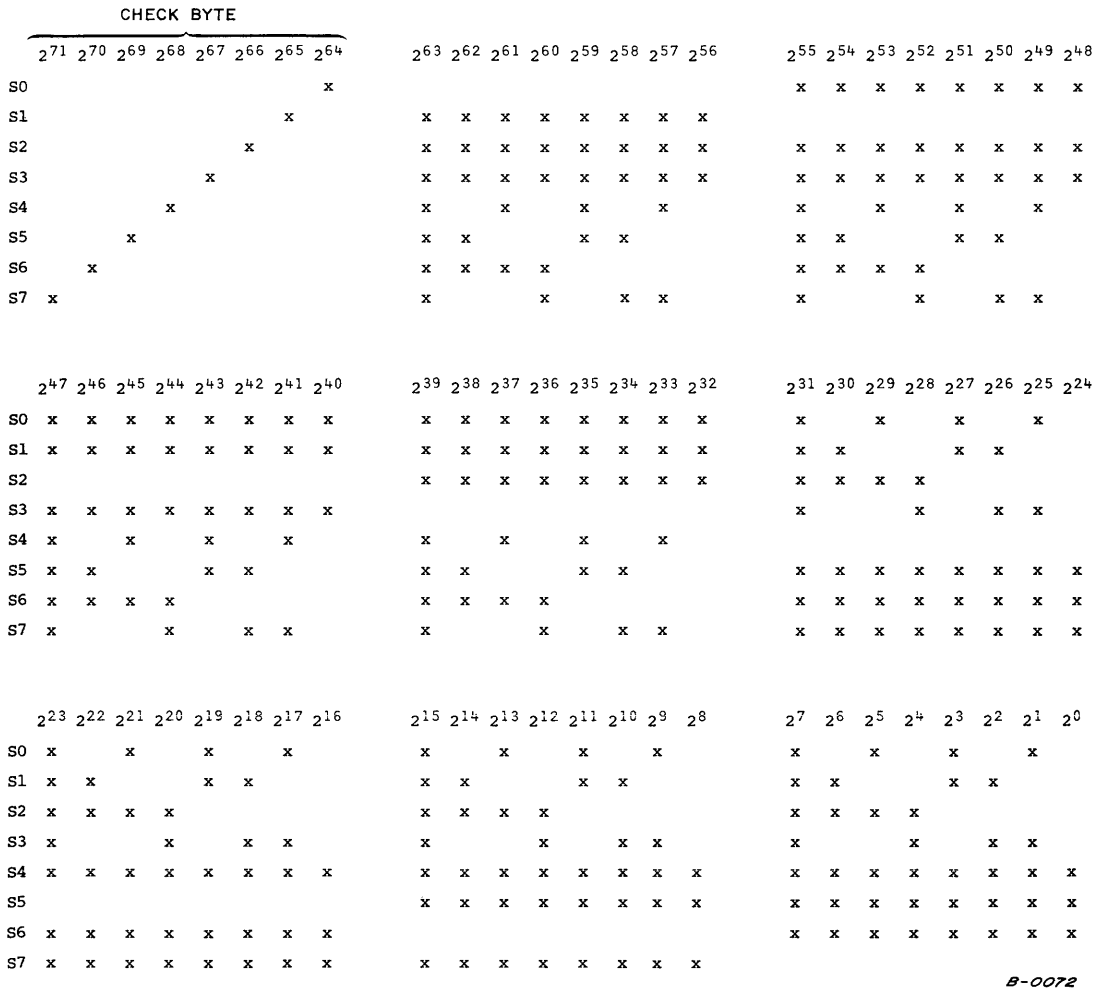


Figure 2-4. Error correction matrix

CPU CONTROL SECTION 3

INSTRUCTION ISSUE AND CONTROL

This section describes the instruction buffers and registers involved with instruction issue and control. Figure 3-1 illustrates the general flow of instruction parcels through the registers and buffers.

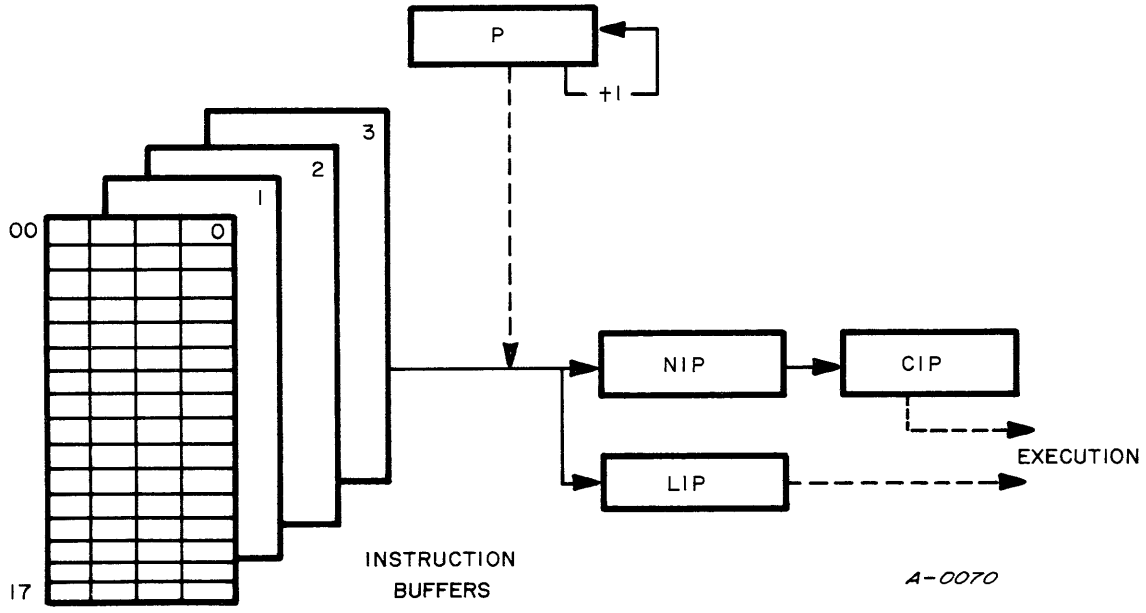


Figure 3-1. Instruction issue and control elements

P REGISTER

The P register is a 24-bit register which indicates the next parcel of program code to enter the next instruction parcel (NIP) register in a linear program sequence. The high-order 22 bits of the P register indicate the word address for the program word in memory. The low-order 2 bits indicate the parcel within the word. The contents of the P register are normally advanced by 1 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. The contents of P are 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 indeterminate value is stored in the terminating exchange package at address 0 during the deadstart sequence.

NIP REGISTER

The NIP (next instruction parcel) register is a 16-bit register that holds a parcel of program code prior to entering the CIP register. A parcel of program code that 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.

CIP REGISTER

The CIP (current instruction parcel) register is a 16-bit register that holds the instruction waiting to issue. If this instruction is a 2-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. An undetermined instruction will issue during the master clear sequence.

LIP REGISTER

The LIP (lower instruction parcel) register is a 16-bit register that holds the lower half of a 2-parcel instruction at the time the 2-parcel instruction issues from the CIP register.

INSTRUCTION BUFFERS

The CPU has four instruction buffers, each of which holds 64 consecutive 16-bit instruction parcels (figure 3-2). Instruction parcels are held in the buffers prior to being delivered to the NIP or LIP registers.

The beginning instruction parcel in a buffer always has a word address that is a multiple of 20_8, (that is, a parcel address that is a multiple of 100_8). This allows the entire range of addresses for instructions in a buffer to be defined by the high-order 18 bits of the beginning parcel address. For each buffer, there is an 18-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 2-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.

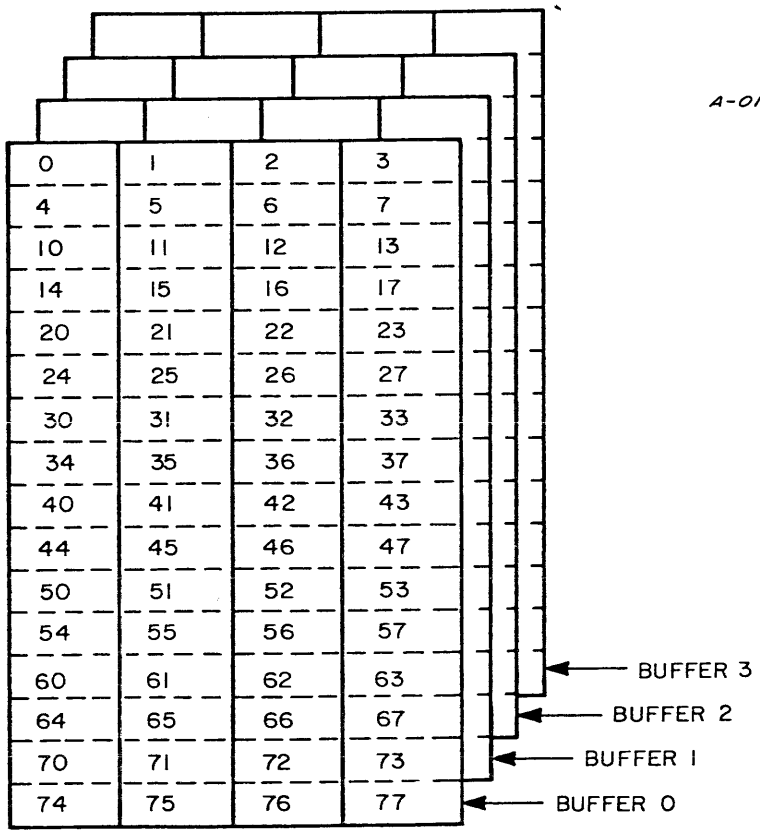


Figure 3-2 Instruction buffers

On an in-buffer conditon, if the instruction is in a different buffer than the previous instruction, a change of buffers occurs, normally necessitating a 2-CP 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 conditon 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 2-bit counter. Each occurrence of an out-of-buffer condition causes the counter to be incremented by 1 so that the buffers are selected in rotation.

Buffers are loaded from memory four words per CP, 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 an apparent constant 11 CPs for 16-bank memories and 15 CPs for 8-bank memories.

An instruction buffer is loaded with one word of instructions from each of the 16 memory banks or two words from each of 8 banks. The first four instruction parcels residing in an instruction buffer are always from bank 0.

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 the buffers to be loaded as needed.

Both forward and backward branching is possible within the buffers. A branch does not cause reloading of an instruction buffer if the instruction 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 CPU, 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, a main program sequence in one or two of the buffers could make 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.

EXCHANGE MECHANISM

Exchange mechanism refers to the technique employed in the CPU for switching instruction execution from program to program. This technique involves the use of blocks of program parameters known as exchange packages and a CPU operation referred to as an exchange sequence.

Throughout the discussion of the exchange package, an alternate bit position representation is used. The bits are numbered from left to right with bit 0 assigned to the 2^{63} bit position. This notation is for the convenience of CAL programmers.

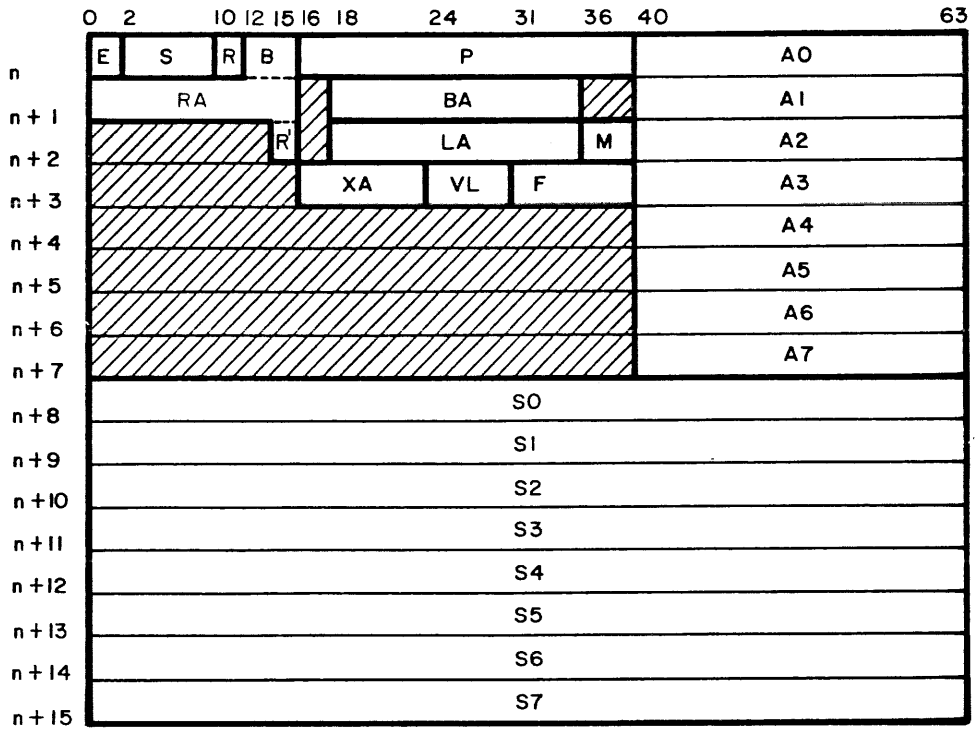
EXCHANGE PACKAGE

An exchange package (figure 3-3) 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) - 24 bits
Base address register (BA) - 18 bits
Limit address register (LA) - 18 bits
Mode register (M) - 4 bits
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. 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 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 program which supervises the object program execution, or by any program that needs this data.



Registers

S Syndrome bits

R'RAB Read address for error

2^{21} 2^{19} 2^{0}

R'	RA	B

P Program address, 24 bits

BA Base address, 18 bits

LA Limit address, 18 bits

XA Exchange address, 8 bits

VL Vector length, 7 bits

E - Error type (bits 0,1)

10 Uncorrectable memory

01 Correctable memory

R - Read mode (bits 10,11)

00 Scalar

01 I/O

10 Vector

11 Fetch

Bit M - Modes

36 Interrupt on correctable memory error

37 Interrupt on floating-point

38 Interrupt on uncorrectable memory error

39 Monitor mode

F - Flags

31 PCI interrupt

32 MCU interrupt

33 Floating-point error

34 Operand range

35 Program range

36 Memory error

37 I/O interrupt

38 Error exit

39 Normal exit

Figure 3-3. Exchange Package

Memory error data

Two bits in the M (mode) register determine whether or not the exchange package contains data relevant to a memory error if one occurs prior to an exchange sequence. The bits 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 memory 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.

Syndrome (S) - The 8 syndrome bits used in detecting a memory data error are returned in bits 2 through 9 of the first word of the exchange package. Refer to section 2 for additional information.

Read mode (R) - This field indicates the read mode in progress when a memory data 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 (includes memory references with A, B, S, or T registers, or exchange sequence)

01 I/O

10 Vector

11 Instruction fetch

These bits are not valid for range errors.

Read address (R'RAB) - The R'RAB field contains the address at which a memory data 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. Bits 2^{14} and 2^{15} of the third word of the exchange package (R') contain bits 2^{21} (or 0) and bit 2^{20} of the address.

EXCHANGE REGISTERS

Three special registers are instrumental in the exchange mechanism: 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 high-order 8 bits of a 12-bit field that specifies the address. The low-order bits of the field are always 0; an exchange package must begin on a 16-word boundary. The 12-bit limit requires that the absolute address be in the lower 4096 ($10,000_8$) 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 the beginning address (XA) in memory.

M Register

The M (mode) register is a 4-bit register that contains part of the exchange package for a currently active program. Bits are assigned as follows in word 2 of the exchange package:

Bit 36 Correctable memory error mode flag. When this bit is set, interrupts on correctable memory data errors are enabled.

Bit 37 Floating-point error mode flag. When this bit is set, interrupts on floating point errors are enabled.

Bit 38 Uncorrectable memory error mode flag. When this bit is set, interrupts on uncorrectable memory data errors are enabled.

Bit 39 Monitor mode flag. When this bit is set, all interrupts other than memory errors are inhibited.

The 4 bits are selectively set during an exchange sequence. Bit 37, the floating-point error mode flag, can be set or cleared during the execution interval for a program through use of the 0021 (EFI) and 0022 (DFI) instructions. The remaining bits are not altered during the execution interval for the exchange package and can be altered only when the exchange package is inactive in storage.

F Register

The F (flag) register is a 9-bit register that contains part of the exchange package for the currently active program. This register contains nine flags which are individually identified within the exchange package. Setting any of these flags causes interruption of 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 bit is set, another exchange occurs immediately.

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 low-order bit of the M register is 0. With the exception of the memory error flag, if the program is in monitor mode and the conditions for setting an F register are otherwise present, the flag remains cleared and no exchange sequence is initiated.

ACTIVE EXCHANGE PACKAGE

An active exchange package is an exchange package that 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. Deadstart sequence
2. Interrupt flag set
3. Program exit

Initiated by deadstart sequence

The deadstart sequence forces the exchange address register content to 0 and also forces a 000 code in the NIP register. These two actions cause the execution of a program error exit using memory address 0 as the location of the exchange package. The inactive exchange package at address 0 is then moved into the operating registers and a program is initiated using these parameters. The exchange package swapped out to address 0 is largely indeterminate as a result of the deadstart operation and is in effect 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 interrupt flags in the F register. One or more flags set results in a request interrupt signal which initiates an exchange sequence.

Initiated by program exit

Two program exit instructions cause the initiation of an exchange sequence. The timing of the instruction execution is the same in either case and the difference is only in which of the two flags in the F register is set. The two instructions are:

Program code 000 ERR - Error exit

Program code 004 EX - 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.

Each of these instructions has a flag in the F register. 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 sensed 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 an instruction other than 000 or 004 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 6 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:

50 CPs; consists of an exchange sequence (36 CPs) and a fetch operation (14 CPs)

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 deadstart 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 ensures 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 deadstart. No interrupts can terminate its execution interval since it is in monitor mode. The monitor program voluntarily exits by issuing a 004 EX 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 exchange address in B's exchange package to point back to the monitor.

The exchange sequence to B causes the exchange 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.

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-processing program could execute in monitor mode or in user mode.

Further guidance on exchange package management is contained in the Cray Research, Inc. CRAY-1 Operating System maintenance documentation.

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 one less than a multiple of 16. The field limits are contained in two registers, the base address register (BA) and the limit address register (LA), described later.

All memory addresses contained in the object program code are relative to the base address which begins the defined field. An object program cannot read or alter any memory location with an absolute address lower than the base address. Each object program reference to memory is checked against the limit and base addresses to determine if the address is within the bounds assigned. A memory write reference beyond the assigned field limits is allowed to issue, but no write occurs. A memory read reference beyond the assigned field limits issues and completes, but data consisting of all zeros is transferred to the appropriate registers.

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 high-order 18 bits of a 22-bit memory address. The low-order 4 bits of the address are assumed 0. Absolute memory addresses are formed by adding the product of 2^4 x (BA) 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 high-order 18 bits of a 22-bit memory address. The low-order 4 bits of the address are assumed 0. 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).

PROGRAM RANGE ERROR

The program range error flag sets if an out-of-range memory reference was for an instruction fetch. This could occur in a non-monitor mode program on a branch or jump instruction that calls for a program address that is above or below the limits. The program range error flag causes an error condition that terminates program execution. The monitor program should check the state of the program range error flag and take appropriate action, perhaps aborting the user program.

OPERAND RANGE ERROR

The operand range error flag sets if an out-of-range memory reference was called to read or write an operand for an A, B, S, T, or V register. The operand range error flag causes an error condition that terminates the user program execution. The monitor program should check the state of the operand range error flag and take appropriate action, perhaps aborting the user program.

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  RT  Enter the real-time clock register with (Sj)

072ixx  Si  RT  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

A programmable clock is incorporated to measure the duration of intervals accurately. A periodic interrupt can be generated with intervals selected under monitor program control. The clock frequency is 80 Mhz. Intervals from 12.5 nanoseconds to about 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

Supporting the programmable clock are four monitor mode instructions and two additional 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 request

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 and the interrupt countdown (ICD) counter when the 0014j4 instruction is executed.

This value is held in the II register and is transferred to the ICD each time the counter reaches 0 and generates an interrupt request. The content of the II register is changed only by another 0014j4 instruction.

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 instruction is executed. This counter runs continuously but counts down, decrementing by 1 each clock period until the content of the counter is 0. 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 programmable 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 programmable clock interrupt instruction.

Following any deadstart, the monitor program should ensure the state of the programmable clock interrupt by clearing programmable clock interrupt requests (0014j5) and disabling programmable clock interrupt requests (0014j7).

DEADSTART SEQUENCE

The deadstart sequence is that sequence of operations that starts a program running in the CPU after power has been turned off and then turned on again or whenever a new system is to be re-initialized in the CPU. 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 following sequence of operations to begin a program is initiated by the MCU or the I/O Subsystem.

1. Turn on master clear signal.
2. Turn on I/O clear signal.
3. Turn off I/O clear signal.
4. Load memory via MCU channel.
5. Turn off master clear signal.

The master clear signal halts 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 MCU channel and activates the MCU input channel. All other input channels remain inactive. The MCU or I/O Subsystem then loads an initial exchange package and monitor program. The exchange package must be located at address 0 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.

CPU COMPUTATION SECTION 4

INTRODUCTION

The CPU section consists of operating registers and functional units which are associated with three types of processing: address, scalar, and vector. For address processing, there are two levels of 24-bit registers and two integer arithmetic functional units. For scalar processing, there are two levels of 64-bit scalar registers, four functional units dedicated solely to scalar processing and three floating-point units shared with the vector operations. For vector processing, there are a set of 64-element registers of 64 bits each, four functional units dedicated solely to vector applications, and three floating-point functional units supporting both scalar and vector operations.

Vector and scalar processing are performed on data while address processing 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 transmsitted to memory from the address registers.

OPERATING REGISTERS

Operating registers are a primary programmable resource of the CPU. 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 CPU has five sets of registers: three primary sets and two intermediate sets.

The three primary sets of registers are address, scalar, and vector designated in this manual as A, S, and V, 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 reference instructions required for scalar and address operands can be greatly reduced. The intermediate registers that support scalar registers are referred to as T registers. The intermediate registers that support the address registers are referred to as B registers.

ADDRESS REGISTERS

The two types of address registers (figure 4-1) are designated A registers and B registers.

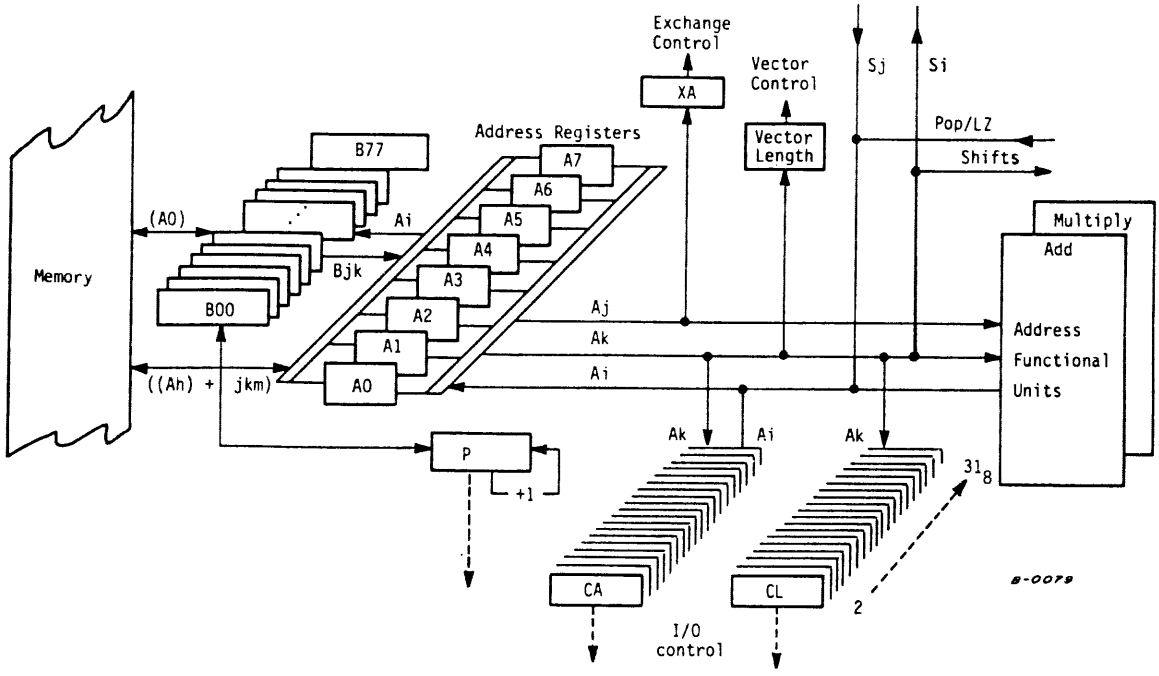


Figure 4-1. Address registers and functional units

A REGISTERS

The eight 24-bit A registers serve a variety of applications, but are primarily used as address registers for memory references and as index registers. They are used to provide values for shift counts, loop control, and channel I/O operations, and also to receive values of population count and leading zeros count. 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 increment 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 by delivering the results to A registers. There are several address adders devoted exclusively to calculations for memory references. These are not available to the program.

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 and XA register are 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 have caused 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 delivers new data to an A register, a reservation is set for that register to prevent issue of instructions that use 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 specifying the subscript as the h, i, j, or k designator as described in part 2, section 6.

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:

```
010ijkm   JAZ
011ijkm   JAN
012ijkm   JAP
013ijkm   JAM

034ijk    Bjk,Ai ,A0
035ijk    ,A0 Bjk,Ai
036ijk    Tjk,Ai ,A0
037ijk    ,A0 Tjk,Ai
```

Refer to part 2, section 6 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 contain data to be referenced repeatedly over a sufficiently long span so that it is not 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 1 CP. 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 specifying the B register number in the jk designator as described in part 2, section 6.

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 (P) and a branch to an address specified by ijkm occurs. Upon receiving control, the called routine conventionally saves (B_{00}) so that the B_{00} register is 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 005000 instruction. This instruction, which is a branch to (B00), causes the address saved in B00 to be entered into P as the address of the next instruction parcel to be executed.

SCALAR REGISTERS

The two types of scalar registers (figure 4-2) are designated S registers and T registers.

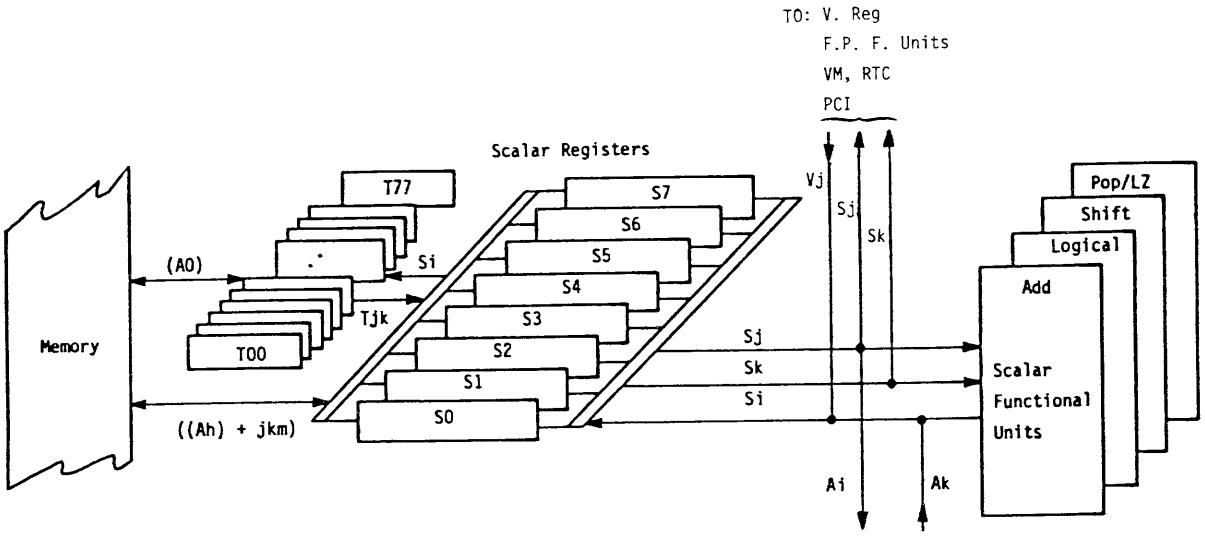


Figure 4-2. Scalar registers and functional units

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 transmission of data between an S register and an element of a V register is 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.

Other uses of the S registers are the setting or reading of the vector mask (VM) register or the real-time clock (RTC) register, or setting the interrupt interval (II) register.

At most, one S register can receive data during each clock period. Issue of an instruction is delayed if it would have caused 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 delivers new data to an S register, a reservation is set for that register to prevent issue of instructions that use 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 specifying the subscript as the i, j, or k designator as described in part 2, section 6.

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.

014ijkm	JSZ
015ijkm	JSN
016ijkm	JSP
017ijkm	JSM

Refer to part 2, section 6 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 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 1 CP. 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 specifying the octal subscript as the jk designator as described in part 2, section 6.

VECTOR REGISTERS

Figure 4-3 illustrates the registers and functional units used for vector operations.

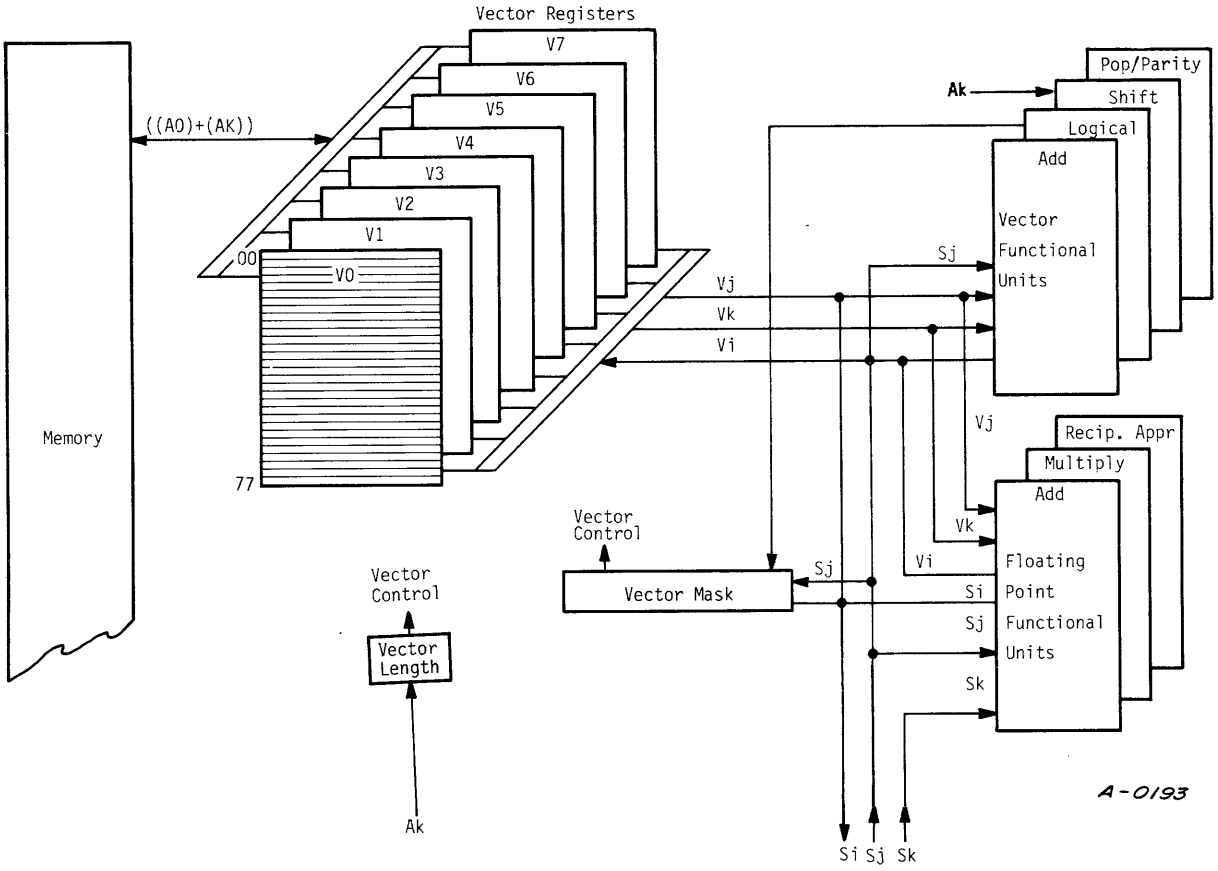


Figure 4-3. Vector registers and functional units

V REGISTERS

Eight V registers, each with 64 elements are the major computational registers of the CPU. 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 treated as a vector. Examples of vector quantities are rows or columns of a matrix or elements of a table.

Computational efficiency is achieved by identically processing each element of a vector. 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 content of the vector length (VL) register.

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.

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. Chained operation is detected automatically by the CPU and is not explicitly specified by the programmer, although the programmer may reorder certain code segments in order to enable chained operation.

A conflict may occur 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 occupies the selected functional unit until the vector is processed.

Parallel vector operations may be processed in two ways:

1. Using different functional units and all different V registers

2. Using the result stream from one vector register simultaneously as the operand to another operation using a different functional unit (chain mode)

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.

The contents of a V register are transferred to or from memory in a block mode by specifying a first word address in memory, an increment or decrement for the memory address, 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 upon 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 postscript in the range 0 through 7. Vector instructions reference V registers by specifying the postscript as the i, j, or k designator as described in section 6 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 2 CPs. 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 normally be changed between operations that chain because chaining implies operations of the same length.

The A0 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.

A vector read cannot chain if speed control is in effect. Speed control is caused by bank conflict due to the increment, which varies between 16-bank and 8-bank machines. Speed control is in effect if the memory address increment is a multiple of eight on a 16-bank machine or is a multiple of four on an 8-bank machine.

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. This register controls the number of operations performed for instructions 140 through 177. The VL register may be set to an A register value using the 0020 instruction.

CAUTION

Cray Research cautions users against increasing VL between operations that may chain together. In some 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 or other 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 instrucion or may be created by testing a vector register for a condition using the 175 instrucion. The mask controls element selection in the vector merge instructions (146 and 147). The contents of VM may be sent to an S register with an 073 instruction.

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 with the exception of the Reciprocal Approximation and Vector Population Count units, which share some logic. 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 3-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 until completion of the calculation is termed the functional unit time and is measured in 12.5-nanosecond clock periods.

The functional units are fully segmented. This means that a new set of operands for unrelated computation may enter a functional unit each clock period even though the functional unit time may be more than 1 CP. This segmentation is possible by capturing and holding the information arriving at the unit or moving within the unit at the end of every clock period.

Thirteen 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, supports either scalar or vector operations and accepts operands from or delivers results to S or V registers accordingly. In addition, for vector operations, memory acts like a fourteenth functional unit.

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 twos 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. The twos complement subtraction for the 031 instruction occurs as follows. The ones complement of the Ak operand is added to the Aj operand. Then a 1 is added in the low order bit position of the result.

No overflow is detected in the functional unit.

The result register reservation time is 2 CPs.

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 least significant 24 bits of the product.

This functional unit is designed to handle address manipulations that do not exceed its data capabilities. Therefore, the programmer must be careful when multiplying integers in the unit, because the unit does not detect overflow of the product and significant portions of the product could be lost.

The result register reservation time is 6 CPs.

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 in the section, Floating-Point Functional Units.

Scalar add unit

The scalar add unit performs 64-bit integer addition and subtraction. It executes instructions 060 and 061. The addition and subtraction are performed in a similar manner. The twos complement subtraction for the 061 instruction occurs as follows. The ones complement of the Sk operand is added to the Sj operand. Then a 1 is added in the low-order bit position of the result.

No overflow is detected in the unit.

The result register reservation time is 3 CPs.

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.

All A register shift counts are considered positive, unsigned integers. If any bit higher than 2^5 is set, the shifted result is all zeros.

The scalar shift unit executes instructions 052 through 057. Single-register shift instructions, 052 through 055, are executed in 2 CPs. Double-register shift instructions, 056 and 057, are executed in 3 CPs.

Scalar logical unit

The scalar logical unit performs bit-by-bit manipulation of 64-bit quantities obtained from S registers. It executes instructions 042 through 051, the mask and Boolean instructions. The 042-051 instructions execute in 1 CP.

Scalar population/parity/leading zero unit

This functional unit executes instructions 026 and 027. Instruction 026ij0 counts the number of bits in an S register having a value of 1 in the operand and executes in 4 CPs. The 026ij1 instruction returns a 1-bit population parity count of the Sj register's contents. Instruction 027, which counts the number of bits of 0 preceding a 1 bit in the operand, executes in 3 CPs. For these instructions, the 64-bit operand is obtained from an S register and the 7-bit result is delivered to an A register.

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 and the population/parity unit, which require only one operand, are exceptions. Results from a vector functional unit are delivered to a V register.

Successive operand pairs are transmitted from a vector register to a functional unit each clock period. The corresponding result arrives at a vector register n+2 CPs 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.

Four 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.

The functional unit is reserved for (VL)+4 CP.

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 executes instructions 154 through 157. The addition and subtraction are performed in a similar manner. For the subtraction operations, 156 and 157, the Vk operand is complemented prior to addition and during the addition a 1 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 3 CPs; the chain slot time is 5 CPs.

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.

All shift counts are considered positive unsigned integers. If any bit higher than 2^5 is set, the shifted result is all zeros.

The vector shift unit executes instructions 150 through 153.

Functional unit time is 4 CPs; chain slot time is 6 CPs.

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 2 CPs, chain slot time is 4 CPs.

Vector population/parity unit

The vector population count unit counts the 1 bits in each element of the source vector register. The total number of 1 bits is the population count. This population count may be an odd or an even number, as shown by the low-order bit of the population count.

The 174ij1 instruction (vector population count) and the 174ij2 (vector population count parity) use the same operation code as the vector reciprocal approximation instruction. Therefore, some of the restrictions for the reciprocal approximation unit also apply for the vector population instructions. The vector population count instruction delivers the total population count to elements of the destination V register.

The vector population count parity instruction delivers the low-order bit of the count to the destination V register.

The functional unit time is 6 CPs; chain slot time is 8 CPs.

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 + 2 CP).

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 handling of element counters. 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 0. Operand registers begin incrementing while the element counter for the result register is held at 0 until functional unit time +2 CP. However, when an operand register is the same as the result register, the element counter for the operand/result register is held at 0 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 1 each clock period.

Note that until functional unit time + 2, the initial contents of element 0 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 1 on each successive clock period, sending the contents of elements 0, 1, 2, ... on successive clock periods.

Thus, the first functional unit time + 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 functional unit time + 2 elements then provide one of the operands used in calculating the results for the next functional unit time + 2 elements. The third group contains results based on the results delivered to the second group, and so on until the final group of functional unit time + 2 elements is generated as determined by the vector length.

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 may depend on the operation being performed. For example, when using the floating-point multiply unit recursively, element 0 of the operand/result register is usually set to an initial value of 1.0.

As an example, consider the summation of a vector of floating-point numbers with the following initial conditions for the vector operation:

- All elements of register V1 contain floating-point values.

- Register V2 provides one set of operands and receives 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 CPs causing sums to be generated in groups of eight (functional unit time + 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.

```
(V200) =                  = (V200)                                                
```

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 approximation unit, which requires only one input operand, is an exception.

A particular floating-point unit is reserved during the entire execution of the vector instruction.

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

Floating-point add unit

The floating-point add unit performs addition or subtraction of 64-bit operands in floating-point format. The unit executes instructions 062, 063, and 170 through 173.

A result is normalized even if the operands are unnormalized.

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

Functional unit time is 6 CPs; chain slot time is 8 CPs.

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 either input operand is 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 and is not considered out-of-range. This provides a fast method of computing a 48-bit integer product.

Functional unit time is 7 CPs; chain slot time is 9 CPs.

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 174ij0. Since the Vector Population Count Unit shares some logic with this unit, the k designator must be 0 for the reciprocal approximation instruction to be recognized.

The input operand is assumed to be normalized and if so the result is normalized. The high-order bit of the coefficient is not tested but is assumed to be a 1. If it is not a 1, the result will be incorrect.

Functional unit time is 14 CPs; chain slot time is 16 CPs.

ARITHMETIC OPERATIONS

Functional units in the CPU perform either twos-complement integer arithmetic or floating-point arithmetic.

INTEGER ARITHMETIC

All integer arithmetic, whether 24 bits or 64 bits, is twos complement and is so represented in the registers as illustrated in figure 4-4. The address add unit and multiply units perform 24-bit arithmetic. The scalar add unit and the vector add unit perform 64-bit arithmetic.

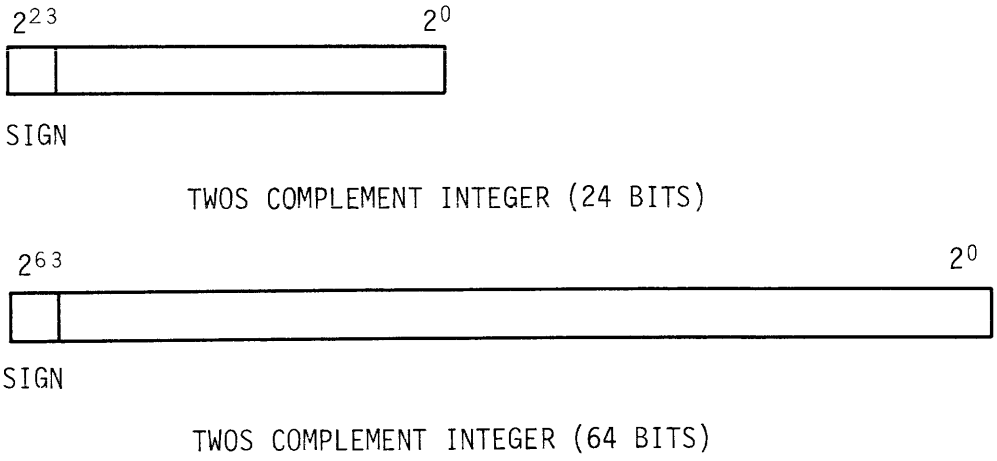


Figure 4-4. Integer data formats

Multiplication of two scaled integer operands may be accomplished by 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 high-order 48 bits of the product of the coefficients as the coefficient of the result and leaves the exponent field zero. See figure 4-6.

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 4-5. Since the coefficient is signed magnitude, it is not complemented for negative values.

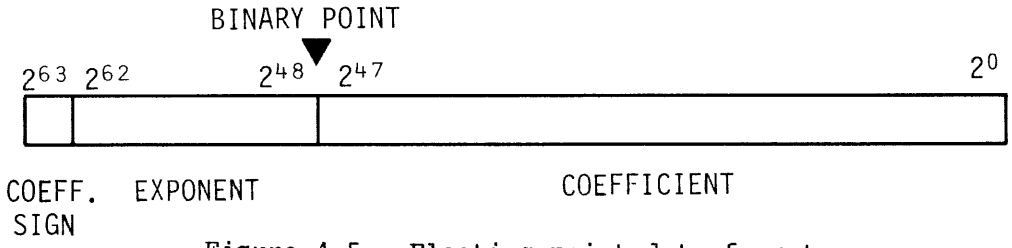


Figure 4-5. Floating-point data formats

The exponent portion of the floating-point format is represented as a biased integer in bits 2^{62} through 2^{48}. 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 0 is not generated by any floating-point functional unit, except in the case of a negative 0 operand going to the floating-point multiply functional unit.

Normalized floating-point numbers

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 0. Since S0 provides a 64-bit zero when used in the Sj field of an instruction, an operand in Sk can be normalized using the 062i0k instruction. Si, which can be Sk, contains the normalized result.

The 170i0k instruction normalizes Vk into Vi if Vk and Vi are different registers.

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 initiates an interrupt if the floating-point mode flag is set in the mode register and monitor mode is not in effect. The floating-point mode flag can be set or cleared by a user mode program.

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

Floating-point add unit - 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
```

NOTE

If the operands to a floating-point add are identical except for sign, the error is suppressed, even though both have exponents greater than or equal to 60000_8.

An underflow condition occurs when the smaller incoming unnormalized exponent is less than or equal to 17777_8. This underflow condition is not detected, and the calculated result goes to the result register.

Floating-point multiply unit - The out-of-range conditions are tested before normalizing. In the floating-point multiply unit, if the exponent of either operand is greater than or equal to 60000_8 or if the sum minus 1 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.

NOTE

If either of the operands is (plus, minus) zero, the error is suppressed, even though the other operand may be out of range.

An underflow condition is detected when the result exponent is less than or equal to 17777_8 and causes an all-zero exponent and coefficient to be returned to the result register.

Underflow is also generated when either, but not both, of the incoming exponents is 0. The condition where both exponents are equal to 0 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 2^{47}. When using this feature, one may consider the operands as 24-bit integers in bits 2^{47} through 2^{24} even though they are actually fractions with the binary point between bits 2^{48} and 2^{47}. In figure 4-6, operand 1 is 4 and operand 2 is 5 which produce a 48-bit result of 24_8. (In this case bit 2^{63} obeys the usual rules for multiplying signs, i.e. - * + = + * - = -, and - * - = + * + = +).

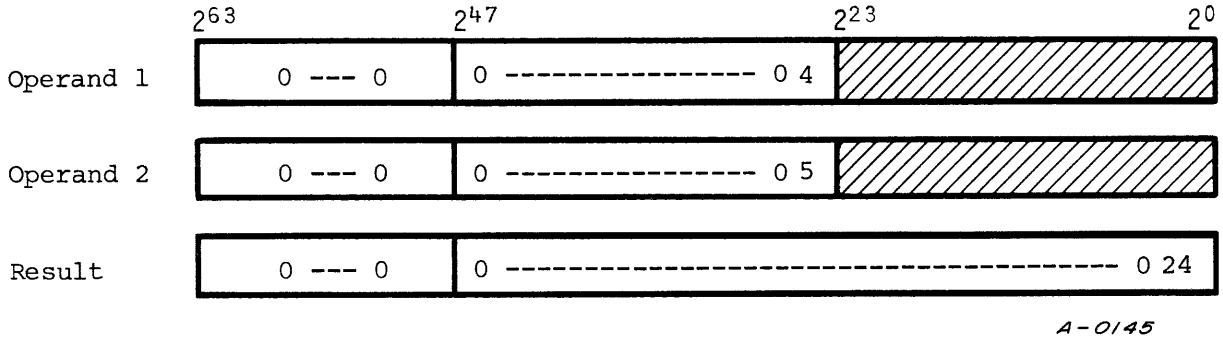


Figure 4-6. Integer multiply in floating-point multiply unit

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 CPU 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 (figure 4-7). 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. The coefficient of the number with the smaller exponent is shifted right to align with the coefficient of the number with larger exponent. Bits shifted out of the register are lost; no round-up takes place. If the sum carries into the high-order bit, the low-order bit is discarded and an appropriate exponent adjustment is made. All results are normalized and if there is underflow, an all zero result is returned.

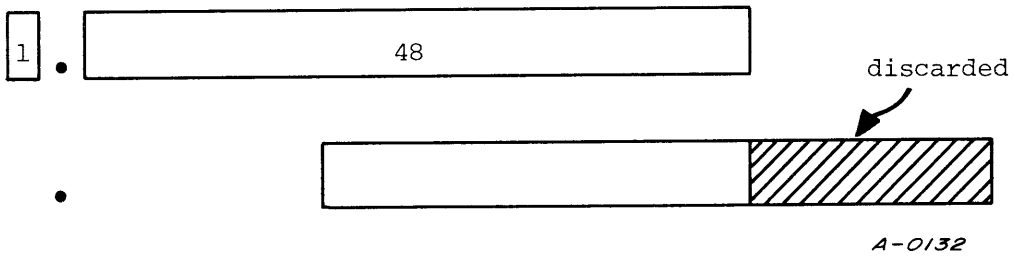


Figure 4-7. 49-bit floating-point addition

Multiplication algorithm

The floating-point multiply unit in the CPU has an input of 48 bits of coefficient into a multiply pyramid (figure 4-8). The pyramid truncates part of the low-order bits of the 96-bit product. To adjust for this truncation, a constant is unconditionally added above the truncation. The value of this constant is $9/2^{56}$. This is an average 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:

$$-0.23 \times 2^{-48} \text{ to } +0.57 \times 2^{-48}$$

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.

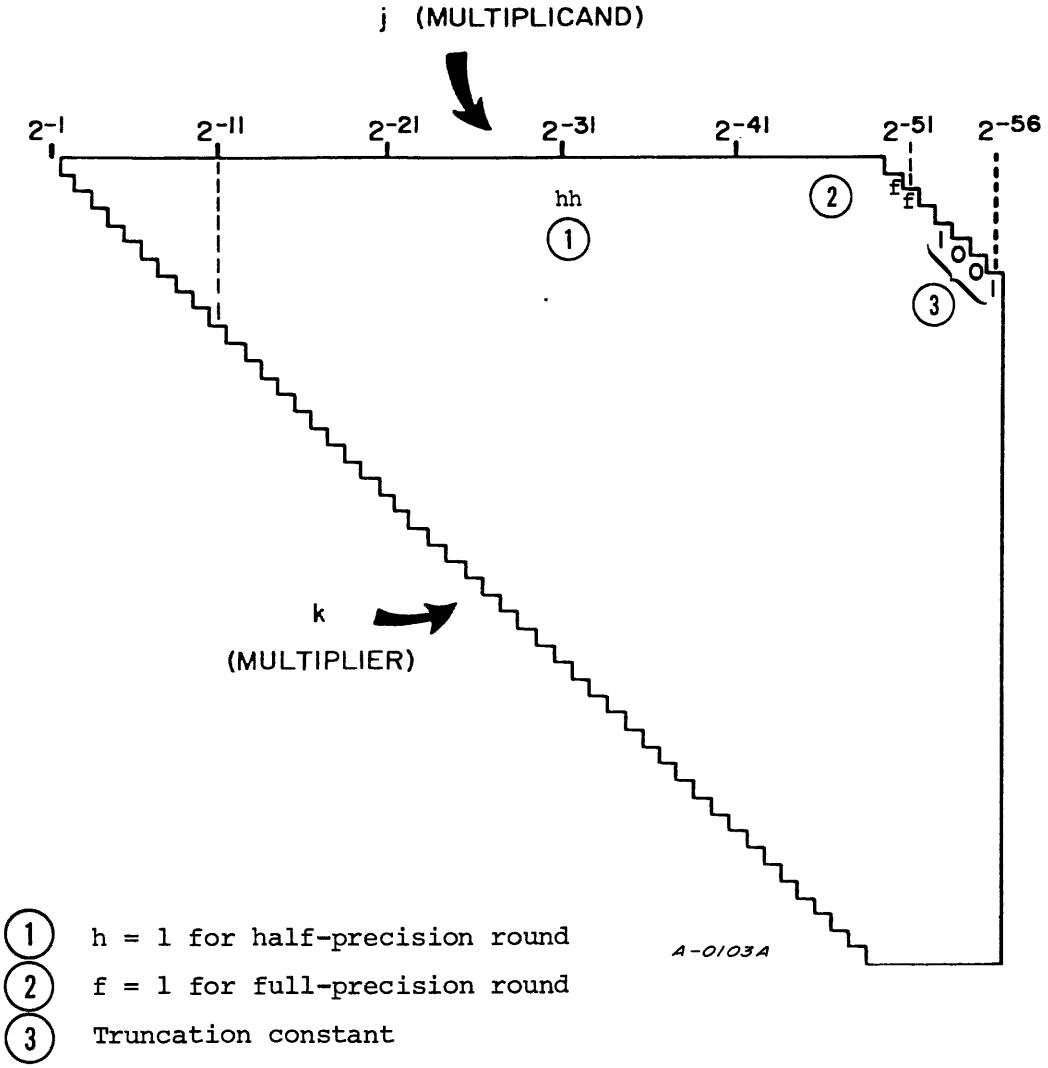


Figure 4-8. Floating-point multiply partial-product sums pyramid

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 29-bit result (2^{-1} to 2^{-29}) 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 1-CP 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 reciprocal approximation is based on Newton's method.

Newton's method - The division algorithm is an application of Newton's method for approximating the real roots of an arbitrary equation $F(x) = 0$, for which $F(x)$ must be twice differentiable with a continuous second derivative. The method requires making an initial approximation (guess), x_0, sufficiently close to the true root, x_t, being sought (see figure 4-9). For a better approximation, a tangent line is drawn to the graph of $y = F(x)$ at the point $(x_0, F(x_0))$. The X intercept of this tangent line is the better approximation x_1. This may be again repeated using x_1 to find x_2, etc.

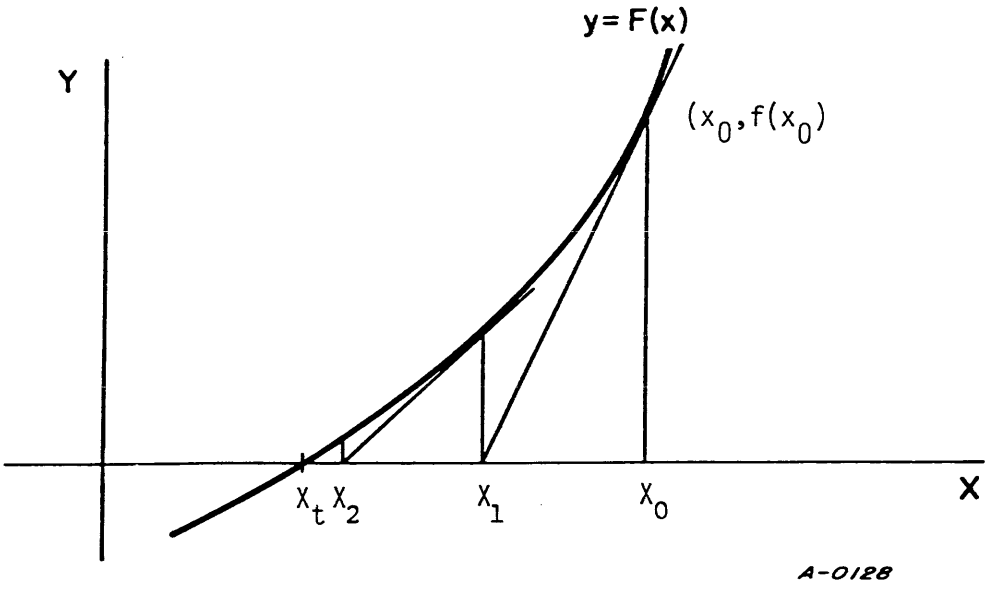


Figure 4-9. Newton's method

Derivation of the division algorithm

A definition for the derivative $F'(x)$ of a function $F(x)$ at point x_t is

$$F'(x_t) = \lim_{x \longrightarrow x_t} \frac{F(x) - F(x_t)}{x - x_t}$$

if this limit exists. If the limit does not exist, $F(x)$ is not differentiable at the point t.

For any point x_i near to x_t,

$$F'(x_t) \approx \frac{F(x_i) - F(x_t)}{x_i - x_t}$$ where $\approx$ means "approximately equal to".

This approximation improves as x_i approaches x_t. Let x_i stand for an approximate solution and let x_t stand for the true answer being sought. The exact answer is the value of x that makes $F(x)$ equal 0. This is the case for $x = x_t$, therefore $F(x_t)$ may be replaced by 0, giving:

$$F'(x_t) \approx \frac{F(x_i)}{x_i - x_t} \qquad (1)$$

Notice that $x_t - x_i$ is the correction that must be applied to an approximate answer, x_i, to give the right answer since $x_i + (x_t - x_i)$ equals x_t. Solving approximation (1) for $(x_t - x_i)$ gives:

$$x_t - x_i = \text{correction} \approx -\frac{F(x_i)}{F'(x_t)}.$$

If this quantity is substituted into the approximation, then:

$$x_t \approx (x_i + \text{correction}) = x_{i+1}.$$

This gives, then, an equation of the form:

$$x_{i+1} = x_i - \frac{F(x_i)}{F'(x_i)}, \qquad (2)$$

where x_{i+1} is a better approximation than x_i to the true value, x_t, being sought. The exact answer is generally not obtained at once because the correction term is not generally exact. However, the operation may be repeated until the answer becomes sufficiently close for practical use.

To make use of Newton's method to find the reciprocal of a number B, simply use $F(x) = (1/x - B)$.

First calculating F'(x) we have:

where $F'(x) = \left(\frac{1}{x} - B\right)' = \left(\frac{-1}{x^2}\right)$. thus for any point $x_1 \neq 0$,

$$F'(x_1) = \frac{-1}{x_1^2} \cdot$$ Choosing for x, a value near $\frac{1}{B}$

and applying equation (2),

$$x_2 = x_1 - \frac{\frac{1}{x_1} - B}{-\frac{1}{x_1^2}},$$

$$x_2 = x_1 + x_1^2 \left(\frac{1}{x_1} - B\right),$$

$$x_2 = x_1 + x_1 - x_1^2 B,$$

$$x_2 = 2x_1 - x_1^2 B = x_1(2 - x_1 B).$$

This approximation technique using Newton's method is implemented in the CRAY-1. A hardware table look up provides an initial guess, x_0, to start the process.

$x_0(2 - x_0B)$	1st approximation, I1	Done in reciprocal unit
$x_1(2 - x_1B)$	2nd approximation, I2	
$x_2(2 - x_2B)$	3rd approximation, I3	
$x_3(2 - x_3B)$	4th approximation	Done with software

The CRAY-1 reciprocal approximation functional unit performs three iterations: I1, I2 and I3. I1 is accurate to 8 bits and is found after a table look-up to choose the initial guess, x_0. I2 is the second iteration and is accurate to 16 bits. I3 is the final (third) iteration answer of the reciprocal functional unit and its result is accurate to 30 bits.

A fourth iteration which uses a special instruction within the floating-point multiply functional unit to calculate the correction term, may be used to increase the accuracy of the reciprocal unit's answer to full precision.

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

1. $S_3 = 1/S_2$ — Performed by the reciprocal approximation unit

2. $S_4 = (2 - (S_3 * S_2))$	Performed by the floating-point multiply unit in iteration mode
3. $S_5 = S_4 * S_3$	Performed by the floating-point multiply unit using full-precision. S_5 now equals $1/S_2$ to 48 bit accuracy.
4. $S_6 = S_5 * S_1$	Performed by the floating-point multiply unit using full-precision rounded

The reciprocal approximation at step 1 is correct to 30 bits. The additional Newton iteration (fourth iteration) at steps 2 and 3 increases this accuracy to 48 bits. This iteration answer is applied as an operand in a full-precision rounded multiply operation to obtain the quotient accurate to 48 bits. Additional iterations should not be attempted since erroneous results could be obtained.

Where 29 bits of accuracy is sufficient, the reciprocal approximation instruction may be used with the half-precision multiply to produce a half-precision quotient in only two operations.

1. $S_3 = 1/S_2$	Performed by the reciprocal approximation unit
2. $S_6 = S_1 * S_3$	Performed by the floating-point multiply unit in half-precision

The 19 low-order bits of the half-precision results are returned as zeros with a round applied to the low-order bit of the 29-bit result.

Another method of computing divisions is as follows:

1. $S_3 = 1/S_2$	Performed by the reciprocal approximation unit
2. $S_5 = S_1 * S_3$	Performed by the floating-point multiply unit
3. $S_4 = (2 - (S_3 * S_2))$	Performed by the floating-point multiply unit
4. $S_6 = S_4 * S_5$	Performed by the floating-point multiply unit

A scalar quotient is computed in 29 CPs since operations 2 and 3 issue in successive clock periods. With this method the full precision reciprocal $1/S_2$ is never formed.

A vector quotient using this procedure requires less than four vector times since operations 1 and 2 are chained together. This overlaps one of the multiply operations. (A vector time is 1 CP for each element in the vector.)

For example, two 60-element vectors are divided in 3 * 60 CPs plus overhead. (The overhead associated with the functional units for this case is 38 CPs).

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 1   1 0 1 0
operand 2   1 1 0 0
result      1 0 0 0
```

A logical difference is the exclusive OR function:

```
operand 1   1 0 1 0
operand 2   1 1 0 0
result      0 1 1 0
```

A logical sum is the inclusive OR function:

```
operand 1   1 0 1 0
operand 2   1 1 0 0
result      1 1 1 0
```

A logical equivalence is the exclusive NOR function:

```
operand 1   1 0 1 0
operand 2   1 1 0 0
result      1 0 0 1
```

The merge uses two operands and a mask to produce results as follows:

```
operand 1   1 0 1 0 1 0 1 0
operand 2   1 1 0 0 1 1 0 0
mask        1 1 1 1 0 0 0 0
result      1 0 1 0 1 1 0 0
```

The bits of operand 1 pass where the mask bit is 1. The bits of operand 2 pass where the mask bit is 0.

CPU INPUT/OUTPUT SECTION 5

INTRODUCTION

The Input/Output section of the Central Processing Unit contains one Memory Channel and 12 I/O channel pairs. The Memory Channel has one input channel and one output channel. The 12 I/O channel pairs are composed of 12 input channels and 12 output channels. The I/O channels are 16 bits wide, while the Memory Channel is 64 bits wide.

MEMORY CHANNEL

The Memory Channel transfers data between the Central Memory and the Buffer I/O Processor (BIOP). It has two independent channels, one for input to Central Memory, and one for output from Central Memory. Each channel is 64 bits wide, and handles data at approximately 850 Mbits per second. Each channel uses an additional 8 check bits for single error correction/double error detection (SECDED), just as is used in Central Memory.

The CPU side of the channel uses a pair of 16-word buffers to stream the data out of Central Memory. As one buffer block is being sent to the I/O Processor, the other buffer is filling from Central Memory. Another pair of buffers is used to speed data into Central Memory.

At the I/O Processor end of the channels, the data passing into I/O Memory is double-buffered and disassembled into 16-bit parcels. The channel end passing data from I/O Memory simply assembles 64-bit words from the 16-bit parcels.

The instruction fetch, exchange sequence, and normal I/O channel memory requests all take precedence over a Memory Channel Central Memory request. Data is sent in blocks, with 16 words as the normal block length. Each block transfer keeps Central Memory busy for 7 CPs and locks out all other memory requests.

Between block transfers there is a 1-CP wait that allows any other active memory requests to take over Central Memory.

The Memory Channel is controlled by the Buffer I/O Processor. There are no CPU instructions for the Memory Channel. All data transfers between the Central Memory and the BIOP are initiated by the BIOP. All error handling is initiated by the BIOP. Since the Memory Channel is supervised by an I/O Processor, the programming details are contained in part 3, section 7.

I/O CHANNELS

The I/O channels have three basic types of control logic:

1. 16-bit asynchronous; used for Maintenance Control Unit interface or front-end interfaces; the standard CRAY-1 I/O channel

2. 16-bit high-speed asynchronous

3. 16-bit synchronous; used for disk storage access

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

CHANNEL GROUPS

I/O channels are numbered 2 through 31_8 and 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)

033i0x Transmit channel number to Ai

033ij0 Transmit address of channel (Aj) to Ai

033ij1 Transmit error flag of channel (Aj) to Ai

BASIC I/O CHANNEL OPERATION

Each input or each output channel directly accesses the Central 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 Central Memory words into 16-bit parcels or 16-bit parcels into 64-bit Central Memory words. Four parcels make up one Central Memory word, with bits of the parcels assigned to memory bit positions as shown in table 5-1. 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 channel current address register (CA), and a channel limit address register (CL).

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 different methods are explained later.

In addition to the three control signals, either the input or output channel of a pair has a Master Clear line.

Table 5-1. 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 channel types on the transmission of the last four parcels:

16-bit asynchronous	Resume for last parcel transmitted sets interrupt
16-bit high-speed asynchronous	Resume for last four parcels transmitted sets interrupt
16-bit synchronous	Interrupt sets when last Ready is sent
16-bit asynchronous	Disconnect received and channel active, or CA = CL and channel active
16-bit high-speed asynchronous	Disconnect received and channel active
16-bit synchronous	Disconnect received and channel active and CA = CL

- External device disconnect received on any input channel and channel is active
- 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 CPU program must perform the following steps:

1. Set the channel limit address to the last word address+1 (LWA+1). See figure 5-1.
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 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.

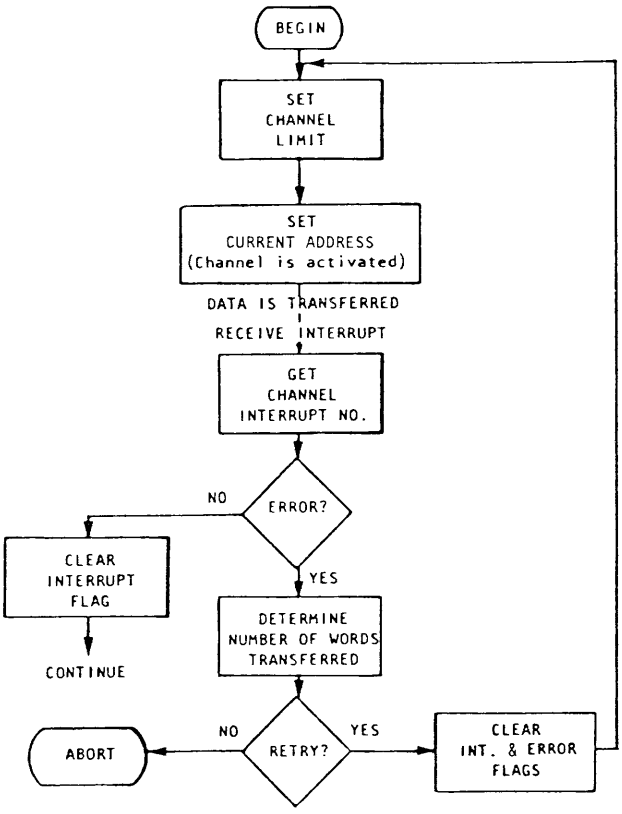


Figure 5-1. Basic I/O program flow chart

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 assembled word. If the partial word is found, the valid portion of the word is stored in memory and the unreceived, low-order parcels are stored as zeros.

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

   16-bit asynchronous channel - When a parcel in error occurs, the parity fault flag sets immediately. The parity fault flag does not generate an interrupt; it is saved and sets the error flag when a disconnect occurs. Therefore, the program should check the state of the error flag when an interrupt is honored.

   16-bit high-speed asynchronous channel - Same as for 16-bit asynchronous channel.

   16-bit synchronous channel - The parcel that contains the error sets the parity fault flag. The parity fault flag causes the subsequent parcels to be stored as all zero parcels. No interrupt is generated by the parity fault. When the data transfer is complete, and the channel goes inactive, the parity fault flag sets the channel error flag. The program should check the state of the error flag when an interrupt is honored.

2. Unexpected Ready pulse

   16-bit asynchronous channel - If a Ready pulse is received when the channel is not active, the ready condition is saved until the channel is activated. At this time a Resume pulse is sent. No error flag is set and no interrupt request is generated.

   If a Ready pulse is received when the memory reference for the previous four parcels is not yet complete, or is received when the channel is active but CA = CL (an extra Ready), the error flag is set. An interrupt request is generated, but no Resume is sent and the data is discarded. When servicing the I/O interrupt, if the channel error flag is set and CA is not equal to CL, a programmed master clear sequence (described later in this section) should be executed on the interrupting channel to clear the external device.

   16-bit high-speed asynchronous channel - If an unexpected Ready pulse is received during a memory reference, the normal burst of four Resume pluses is sent and the data is not sampled. The error flag is set and an interrupt is generated. If the channel is not active or CA = CL when the unexpected Ready pulse arrives, no Resume pulses are sent; the data is not sampled; and the error flag is set to generate an interrupt.

16-bit synchronous channel - A Ready signal is not expected when the channel is inactive, or when CA = CL, or after the first Ready but before the end of the transfer. If an unexpected Ready signal is received, the error flag is set and an interrupt is generated. No further data of the block is transferred. No Resume signal is returned in response to the unexpected Ready signal.

OUTPUT CHANNEL PROGRAMMING

To start an output operation, the CPU 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 1 and starts the data transfer.

After each word is read from memory and the current address is advanced, the 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 operation is complete as soon as the last parcel transfer is finished. Tables 5-3, 5-5, and 5-7 show the terminating sequence.

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. No external response is generated.

16-BIT ASYNCHRONOUS CHANNELS

Input channels

Table 5-2 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 the 16-bit parcel of data from the external device to the CPU. 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 5-2. 16-bit asynchronous input channel signal exchange

	CPU		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.	If (CA) = (CL), go to 13.		
11.			If more data, go to 2.
12.		←	Disconnect (ignored if CA = CL or if channel not active
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 CPU 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 CPU indicates that a parcel of data is being sent to the CPU input channel and may be sampled. The Ready signal is a pulse 50 $\pm$10 nanoseconds wide (at 50% voltage points). The leading edge of Ready at the CPU begins the timing for sampling the data bits.

Resume - Resume is sent from the CPU to the external device to show that the parcel was received and that the CPU is ready for the next data transmission. Resume is a pulse 50 $\pm$3 nanoseconds wide (at 50% voltage points).

Disconnect - This signal is sent from the external device to the CPU 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 5-3 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 CPU 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 Resume pulse is received.

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 CPU 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.

Table 5-3. 16-bit asynchronous output channel signal exchange

CPU	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) ≠ (CL), go to 2.	
12. Disconnect ⟶	
13. Set interrupt and deactivate channel.	

Ready - The Ready signal sent from the CPU 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 CPU 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 CPU has received the Resume from the last Ready. The Disconnect is a pulse 50 ±3 nanoseconds wide (at 50% voltage points).

Cabling Restrictions - The normal length of cable for the 16-bit asynchronous channel is 70 feet (21.3 meters). This assumes a 10-foot (305-cm) drop cable at the CPU, a 50-foot (15.3-meter) data cable, and another 10-foot (305-cm) drop cable at the external device. When used with a Cray Research, Inc. front-end interface, the cable length increases to 300 feet (91.5 meters).

16-BIT HIGH-SPEED ASYNCHRONOUS CHANNELS

Input channels

Table 5-4 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 to the CPU. 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 nanoseconds after the leading edge of Ready and held until 50 nanoseconds 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. 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}

Table 5-4. 16-bit high-speed asynchronous input channel signal exchange

	CPU	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.	

Parity bits are sent from the external device to the CPU 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 CPU indicates that data will soon be sent to the CPU 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 CPU begins the timing for sampling the data bits.

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. 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 nanoseconds.

Resume - This signal is sent to the external device to show that the CPU 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 nanoseconds.

Disconnect - This signal is sent from the external device to the CPU 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 nanoseconds 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 5-5 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 CPU to an external device. They are all sent at the same time, within 5 nanoseconds of the leading edge of the Ready pulse. Data bits remain steady on the lines until the next parcel is sent, or until the Resume pulse is received, whichever occurs first.

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 CPU 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 result from 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 CPU 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.

Table 5-5. 16-bit high-speed asynchronous output channel signal exchange

CPU	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. Data 2^{47}-2^{32} with Ready ⟶	
5. Data 2^{31}-2^{16} with Ready ⟶	
6. Data 2^{15}-2^{0} with Ready ⟶ (with Disconnect if this is the last word)	
7.	⟵ Resume
8. If (CA) ≠ (CL), go to 2.	
9. Set interrupt and deactivate channel.	

Resume - Resume is sent from the external device to the CPU 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 CPU no earlier than 230 nanoseconds after the leading edge of the first Ready pulse is transmitted.

Disconnect - Disconnect is a signal sent from the CPU to the external device that means the transmission from the CPU is complete. It is sent with the last Ready ±3 nanoseconds. The Disconnect pulse is 50 ±3 nanoseconds wide (at the 50% voltage points).

Cabling Restrictions - The 16-bit high-speed asynchronous channel can handle a cable length of up to 70 feet (21.3 meters). This assumes a 10-foot (305-cm) drop cable at the CPU, a 50-foot (15.3-meter) data cable, and another 10-foot (305-cm) drop cable at the external device.

Input channels

Table 5-6 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 CPU. 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. Data lines must be stable at the CPU backpanel within 90 nanoseconds after the corresponding Resume pulse is transmitted from the CPU backpanel.

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 CPU 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 $\pm$10 nanoseconds wide (at the 50% voltage points). It is sent from the external device to the CPU. Ready occurs within 5 nanoseconds of the leading edges of the first parcel of data bits.

Resume - Resume is sent from the CPU to the external device to initiate the synchronous data transfer and to time the sending of data at the CPU. The first Resume pulse is 50 $\pm$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 (150 $\pm$7 ns) followed by as many groups of four Resumes (200 $\pm$9 ns) as required to complete the block transfer.

Disconnect - Disconnect is a signal sent from the external device to the CPU 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 $\pm$10 nanoseconds wide (at the 50% voltage points).

Block length restrictions - The input channel has no restrictions on block length. The disk 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.

Table 5-6. 16-bit synchronous input channel signal exchange

CPU		EXTERNAL
1.	Activate channel (set CL and CA).	
2.	Resume ⟶	
3.		⟵ Data 2^{63}-2^{48} with Ready
4.	Resume ⟶ (150 ns pulse)	
5.	Resume ⟶ (150 ns pulse)	⟵ Data 2^{47}-2^{32}, no Ready
6.	Resume ⟶ (150 ns pulse)	⟵ 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 ⟶ (200 ns pulse)	
11.	Resume ⟶ (200 ns pulse)	⟵ Data 2^{63}-2^{48}, no Ready
12.	Resume ⟶ (200 ns pulse)	⟵ Data 2^{47}-2^{32}, no Ready
13.	Resume ⟶ (200 ns pulse)	⟵ 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.	

Clock - A Clock signal is supplied over a separate cable (one per DCU cabinet) to the external device from the CPU. This Clock signal synchronizes signals at the external device interface connector.

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 CPU 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 CPU and one 7 foot (213 cm) length of data cable at the external device.

Output channels

Table 5-7 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 CPU to the external device. They are sent with the leading edge of the Ready pulse $\pm$ 5 nanoseconds. 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 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 CPU 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 is a static signal sent from the CPU 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 CPU to the external device to indicate that the data is valid. The first Ready signal is a pulse 50 $\pm$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 (150 $\pm$7 ns) followed by as many groups of four Readies (200 $\pm$9 ns) as required to complete the block transfer.

Resume - Resume is sent from the external device to the CPU in response to the first Ready signal. The Resume pulse is 50 $\pm$10 nanoseconds wide (at the 50% voltage points).

Table 5-7. 16-bit synchronous output channel signal exchange

CPU	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 ⟶	150 ns Ready pulse
6. Data 2^{31}-2^{16} with Ready ⟶	
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))	200 ns Ready pulse
11. Data 2^{47}-2^{32} ⟶	
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.	

Disconnect - Disconnect is a signal sent from the CPU to the external device indicating that the transmission from the CPU 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 disk 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.

Clock - A Clock signal is supplied over a separate cable (one per DCU cabinet) to the external device from the CPU. This Clock signal synchronizes signals at the external device interface connector.

Cabling restrictions - The 16-bit 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 CPU 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 CPU and one 7-foot (213 cm) length of data cable at the external device.

PROGRAMMED MASTER CLEAR TO EXTERNAL DEVICE

The CPU contains a mechanism for sending a Master Clear signal to an external device. The Master Clear is sent by either the input channel or the output channel as follows:

- Asynchronous channels - Master Clear sent on input channel
- High-speed asynchronous channels - Master Clear sent on output channel
- Synchronous channels - Master Clear sent on output channel

Sequence for asynchronous channels

The external Master Clear sequence for 16-bit normal-speed asynchronous channels is as follows:

1. 0012jk — Clear output channel to ensure that CPU activity on the channel pair has stopped.
2. 0012jk — Clear input channel to ensure that external activity on the channel pair has stopped.
3. 0011jk — Set the input channel limit to an arbitrary value.
4. 0010jk — Set the input channel current address equal to the same value. This initiates the Master Clear signal.
5. 0012jk — 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. 0011jk 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.

For Cray Research front-end interfaces, delays 1 and 2 should both be a minimum of 80 CPs.

Sequence for high-speed asynchronous and synchronous channels

The external Master Clear sequence for high-speed asynchronous and synchronous channels is as follows:

1. 0012jk Clear output channel interrupt to ensure that CPU activity on the channel pair has stopped.

2. 0012jk Clear input channel interrupt to ensure that external activity on the channel pair has stopped.

3. 0011jk Set the output channel limit to an arbitrary value.

4. 0010jk Set the output channel current address equal to the same value. This initiates the Master Clear signal.

5. 0012jk 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. 0011jk 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.

For the synchronous channel, delay 1 should be a minimum of 1 CP and delay 2, a minimum of 20 CPs.

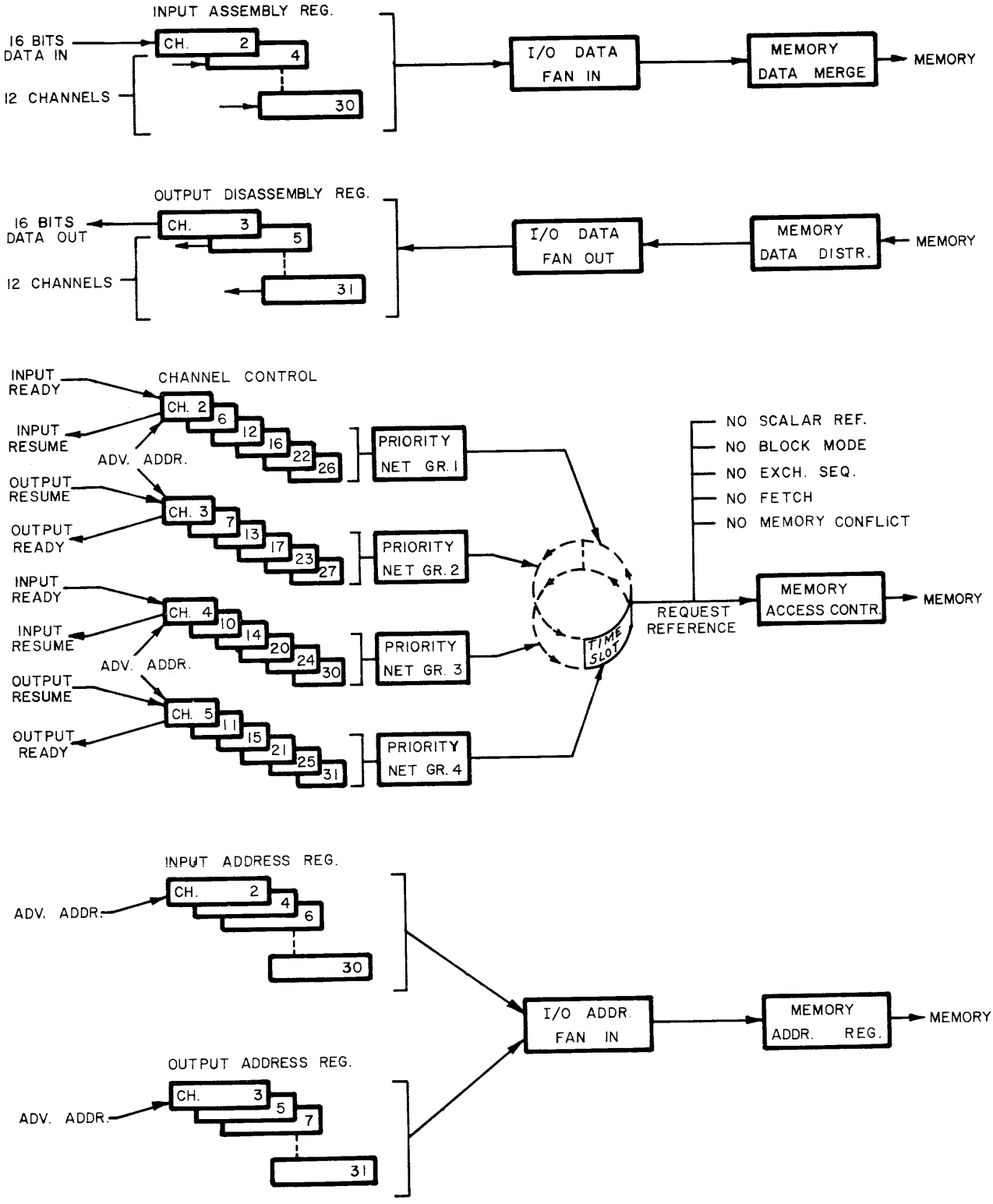


Figure 5-2. Channel I/O control

MEMORY ACCESS

Each of the four channel groups is assigned a time slot (figure 5-2), which is scanned once every 4 CPs 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 once every 8 CPs. However, another channel in the same group as a channel that has just made a memory request can cause a memory request 4 CPs later. During the next 3 CPs, the scanner allows requests from the other three channel groups. Therefore, it is possible to have an I/O memory request every clock period.

I/O LOCKOUT

An I/O memory request can be locked out by a transfer using the B, T, or V registers. Multiple transfers of these types cannot issue without allowing one waiting I/O reference to complete. The maximum duration of a lockout caused by these types of 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) delay 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 4 CPs. To test for a memory bank conflict, the 4 low-order bits[§] of the memory address move through three registers staying 1 CP in each register. The first register is rank A, the second is rank B, and the third is rank C. In the fourth clock period, the address is placed in the memory address register.

I/O MEMORY CONFLICTS

Before testing for a memory bank conflict, a check is made to ensure that no block transfer, exchange sequence, or instruction fetch sequence is in progress, and that no address or scalar instruction requiring a memory reference is in its second clock period of execution. If any of these conditions exist, the I/O request is blocked and must be resubmitted 8 CPs later. The 4 low-order address bits[§] of an I/O reference are tested against address bits in 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 8 CPs before it can request again.

I/O MEMORY REQUEST CONDITIONS

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 address or scalar instruction requiring a memory reference in CP 2 of execution
4. No fetch request
5. No 176, 177 or 034 through 037 in process
6. No exchange sequence
7. No 033 request

I/O MEMORY ADDRESSING

All I/O memory references are absolute. The current address and limit address registers are 22 bits, allowing I/O access to all of memory. Setting of the current address and limit address registers is limited to monitor mode. I/O Memory reference addresses are not checked for range errors.

---

§ Three bits for 8-bank phasing; refer to part 2, section 2 of this manual.

CPU INSTRUCTIONS 6

INSTRUCTION FORMAT

Each instruction is either a 1-parcel (16-bit) instruction or a 2-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 2-parcel instruction may begin in any parcel of a word and may span a word boundary. A 2-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.

Figure 6-1 illustrates the general form of instructions:

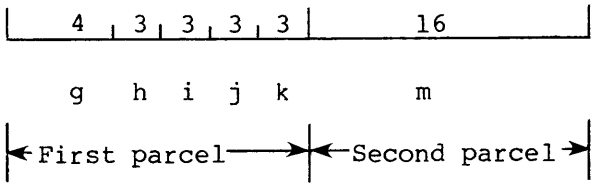


Figure 6-1. General form for instructions

Five variants of this general format use the fields in different ways. Two of these variant forms are 2-parcel formats, two are 1-parcel formats, and one is either a 1-parcel or a 2-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 shown in figure 6-2.

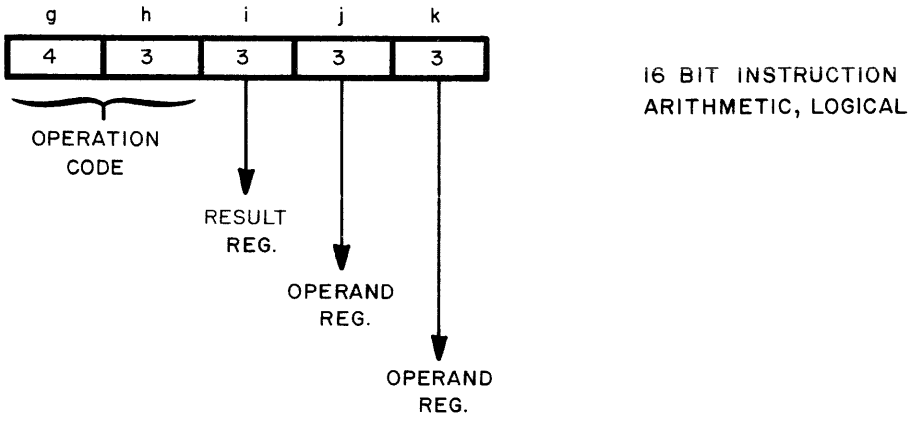


Figure 6-2. Format for arithmetic and logical instructions

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 designates the result and operand registers. The 6-bit combined jk field specifies a shift or mask count. This format is shown in figure 6-3.

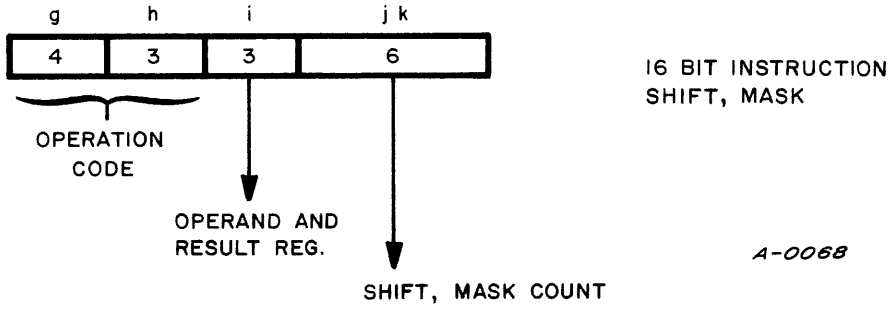


Figure 6-3. Format for shift and mask instructions

IMMEDIATE CONSTANT FORMAT

The instructions that enter immediate constants into A registers have either a 1-parcel or a 2-parcel form. Only the 2-parcel form exists

for entering immediate constants into S registers. For the 1-parcel form, the j and k fields are combined to give a 6-bit quantity. For the 2-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. The instruction format for immediate constant instructions is shown in figure 6-4.

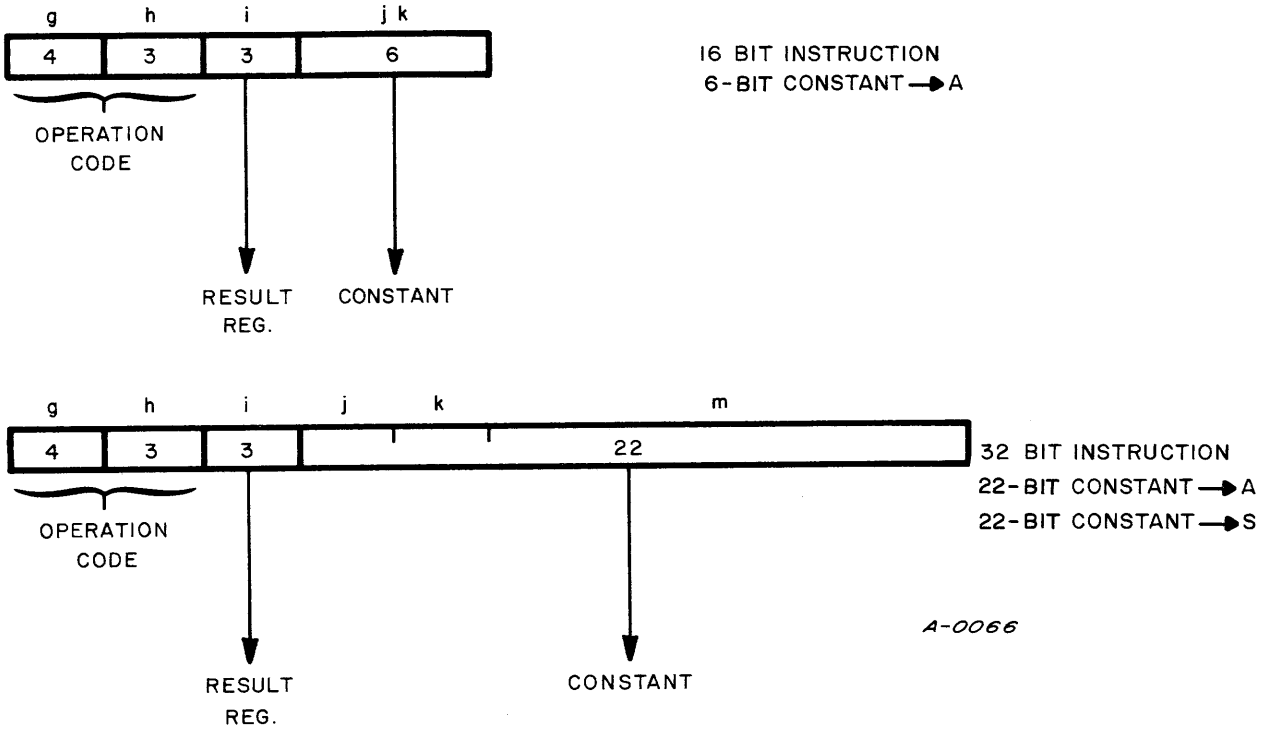


Figure 6-4. Format for immediate constant instructions

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 address field. The first 3-bit field (h) designates an index register (Ah).

When the h field is 0, the special value of 0 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 address displacement value. Figure 6-5 illustrates the format of memory transfer instructions.

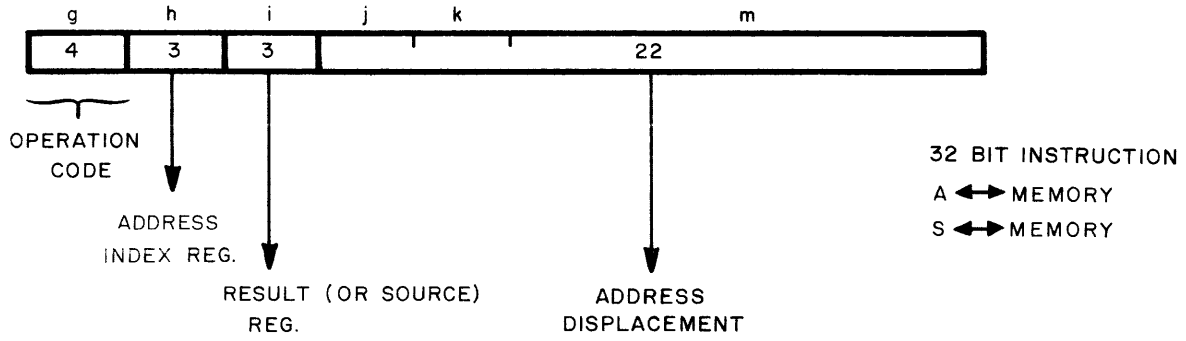


Figure 6-5. Format for memory transfer instructions

BRANCH FORMAT

In general, the branch instructions are 2-parcel instructions. A 7-bit operation code (gh) is followed by a 24-bit field formed by combining i, j, k, and m. The high-order bit of the i field is unused. The 24-bit field contains a parcel address and allows branching to a parcel. Figure 6-6 illustrates the 2-parcel format for branch instructions.

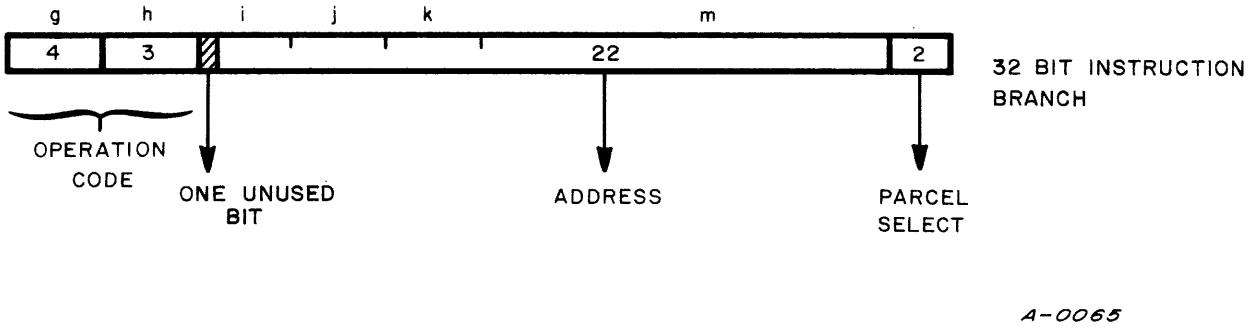


Figure 6-6. Two-parcel format for branch instructions

The unconditional branch to (Bjk) instruction uses only the upper 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 m field is ignored.

SPECIAL REGISTER VALUES

The S_0 and A_0 registers provide special values if 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. The special value is available regardless of existing A_0 or S_0 reservations. This 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, as the operand, the actual value of the register is provided.

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 Next Instruction Parcel (NIP) register. The instruction is passed to the Current Instruction Parcel (CIP) register when the previous instruction issues. An instruction in CIP issues when the conditions in the functional units and registers are such that the functions required for execution may be performed without conflicting with a previously issued instruction. Instruction parcels may issue out of the CIP register at a maximum rate of one per clock period. Once an instruction has been delivered to the CIP, 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, except for scalar memory access instructions (10h and 12h).

Entry to the NIP is blocked for the second half of a 2-parcel instruction. The parcel is delivered to the Last Intruction Parcel (LIP) register, instead. The blank NIP for the second parcel is issued as a do-nothing instruction when it reaches the CIP.

When special register values (A_0 or S_0) are selected by an instruction for Ah, Aj, Ak, Sj or Sk, the normal "hold issue until operand ready" conditions do not apply. These values are always immediately available.

INSTRUCTION DESCRIPTIONS

This section contains detailed information about individual instructions or groups of related instructions. Descriptions are presented with the CRAY-1 Assembler Language (CAL) syntax and 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 CPn, the next instruction may issue at CPn + issue time if its own issue conditions have been met.

CAL Syntax	Description	Octal Code
ERR	Error exit	000000
ERR exp[§]	Programmer coded error exit	000ijk

A 000 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 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 an instruction fetch operation (14 CPs).

Special cases

Inhibit instruction issue

Begin exchange sequence

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
CA,Aj Ak	Set the current address (CA) register for the channel indicated by (Aj) to (Ak) and activate the channel	0010jk
CL,Aj Ak	Set the limit address (CL) register for the channel indicated by (Aj) to (Ak)	0011jk
CI,Aj	Clear the interrupt flag and error flag for the channel indicated by (Aj)	0012jx
XA Aj	Enter the XA register with (Aj)	0013jk

These instructions are privileged to monitor mode and provide operations useful to the operating system. Functions are selected through the i designator. The instructions are treated as pass instructions if the monitor mode bit is not set.

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 first and then setting CA activates the channel. As the transfer continues, CA is incremented toward CL. When (CA) equal (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.

Hold issue conditions

034 - 037 in process

Exchange in process

For 0010 and 0011, Aj or Ak reserved

For 0012 or 0013, Aj reserved

Execution time

Instruction issue, 1 CP

Special cases

If the program is not in monitor mode, the 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 even in monitor mode
If (Aj) is less than 2 or (Aj) is greater than or equal to 31_8, the instruction is a no-op
If k = 0, CA or CL is set to 1

For 0013:
If j = 0, XA register is cleared

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

CAL Syntax	Description	Octal Code
RT Sj	Enter the real-time clock register with (Sj)	0014j0
PCI Sj	Enter interrupt interval (II) register with (Sj)	0014j4
CCI	Clear the programmable clock interrupt request	0014j5
ECI	Enable programmable clock interrupt request	0014j6
DCI	Disable programmable clock interrupt requests	0014j7

These instructions perform specialized functions for managing the real-time and programmable clocks. They are privileged to monitor mode. The instructions are treated as pass instructions if the monitor mode bit is not set.

When the k designator is 0, the 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 low-order 32 bits from the Sj register into both the Interrupt Interval (II) register and the Interrupt Countdown (ICD) counter. When the j designator is 0, II and ICD are cleared.

When the k designator is 5, this instruction clears the programmable clock interrupt request if the request was previously set by an interrupt countdown to 0.

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.

Hold issue conditions

034 - 037 in process

Exchange in process

Sj reserved

Execution time

Instruction issue, 1 CP

Special cases

If the program is not in monitor mode, these instructions become no-ops but all hold issue conditions remain effective.

For 0014j0 and 0014j4, if j=0, (Sj)=0.

The instruction forms 0015, 0016, 0017 are not implementable in the CRAY-1/S hardware but they act as no-op instructions. There is no CAL syntax for them.

CAL Syntax	Description	Octal Code
VL Ak	Transmit (Ak) to VL register	0020xk
VL 1§	Transmit 1 to VL register	0020x0

This instruction enters the vector length (VL) register with a value determined by the contents of Ak. The low-order 7 bits of (Ak) are entered into the VL register. The number of operations performed in a vector instruction is determined by first subtracting 1 from the contents of VL and then adding 1 to the low-order 6 bits of the result.

For example:

if (VL)=100_8	then	$100_8-1 = 77_8$
	and	$77_8+1 = 100_8$
if (VL)=0	then	$0-1 = 177_8$
	and	$77_8 + 1 = 100_8$

Thus, the number of vector operations is 64 when the content of VL is 0 or 64.

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) = Ø if k = 0 and (Ak) = 0

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
EFI	Enable interrupt on floating-point error	0021xx
DFI	Disable interrupt on floating-point error	0022xx

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 range errors as described in section 4.

Hold issue conditions

034 - 037 in process

Exchange in process

Ak reserved

Execution time

Instruction issue 1 CP

Special cases

The 0023, 0024, 0025, 0026 and 0027 instructions are not implemented but act as no-ops. There is no CAL Syntax for them.

CAUTION

The operating system may have status bits that reflect whether interrupts on floating-point range errors are enabled or disabled. If so, those software status bits need to be modified to agree with the floating-point mode flag.

CAL Syntax	Description	Octal Code
VM Sj	Transmit (Sj) to VM register	003xjx
VM 0§	Clear VM register	003x0x

This instruction enters the vector mask (VM) register with the contents of Sj. The VM register is cleared if the j designator is 0. 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

VM ready in 6 CPs for 073 instruction

Special cases

(Sj) = 0 if j = 0

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
EX	Normal exit	004xxx
EX exp[§]	Normal exit, programmer encoded	004ijk

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 an instruction fetch operation (14 CPs)

Special cases

Inhibit instruction issue

Begin exchange sequence

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
J Bjk	Branch to (Bjk)	005xjk

This instruction sets the P register to the 24-bit 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:

Instruction parcel and following parcel both in a buffer and branch address in a buffer, 7 CPs

Instruction parcel and following parcel both in a buffer and branch address not in a buffer, 16 CPs

Parcel following instruction parcel not in a buffer and branch address in a buffer, 16 CPs

Parcel following instruction parcel not in a buffer and branch address not in 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.

CAL Syntax	Description	Octal Code
J exp	Branch to ijkm	006ijkm

This 2-parcel instruction sets the P register to the parcel address specified by the low-order 24 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 instruction in the same buffer and branch address in a buffer, 5 CPs

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

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

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

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

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

Special cases

None

CAL Syntax	Description	Octal Code
R exp	Return jump to ijkm; set B_{00} to (P)	007ijkm

This 2-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 24 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 the B_{00} register.

Hold issue conditions

034 - 037 in process

Exchange in process

Execution time

Instruction issue:

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

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

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

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

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

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

Special cases

None

CAL Syntax	Description	Octal Code
JAZ exp	Branch to ijkm if (A_0) = 0	010ijkm
JAN exp	Branch to ijkm if (A_0) ≠ 0	011ijkm
JAP exp	Branch to ijkm if (A_0) positive	012ijkm
JAM exp	Branch to ijkm if (A_0) negative	013ijkm

These 2-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 24 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 previous 2 CPs

Execution time

Instruction issue:

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

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

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

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

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

010-013

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

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

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

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

Special case

$(A_0) = 0$ is considered a positive condition.

CAL Syntax	Description	Octal Code
JSZ exp	Branch to ijkm if $(S_0) = 0$	014ijkm
JSN exp	Branch to ijkm if $(S_0) \neq 0$	015ijkm
JSP exp	Branch to ijkm if (S_0) positive	016ijkm
JSM exp	Branch to ijkm if (S_0) negative	017ijkm

These 2-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 24 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 previous 2 CPs

Execution time

Instruction issue:

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

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

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

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

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

014-017

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

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

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

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

Special case

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

CAL Syntax	Description	Octal Code
Ai exp	Transmit jkm to Ai	020ijkm
Ai exp	Transmit ones complement of jkm to Ai	021ijkm

The 020 instruction enters into Ai a 24-bit value that is composed of the 22-bit jkm field and 2 high-order bits of 0.

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 2 high-order bits of 0. The complement is formed by changing all 1 bits to 0 and all 0 bits to 1. Thus, for the 021 instruction, the high-order 2 bits of Ai are set to 1 and the instruction provides a means of entering a negative value into Ai. The instructions are both 2-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

Both parcel in different buffers, 4 CPs

Second parcel not in a buffer, 13 CPs

Ai ready, 1 CP

Special cases

None

CAL Syntax	Description	Octal Code
Ai exp	Transmit jk to Ai	022ijk

This 1-parcel instruction enters the 6-bit quantity from the jk field into the low-order 6 bits of Ai. The high-order 18 bits of Ai are zeroed. No sign extension occurs.

Hold issue 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

CAL Syntax	Description	Octal Code
Ai Sj	Transmit (Sj) to Ai	023ijx

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 case

(Sj) = 0 if j = 0

CAL Syntax		Description	Octal Code
Ai Bjk	024ijk	Transmit (Bjk) to Ai	024ijk
Bjk Ai	025ijk	Transmit (Ai) to Bjk	025ijk

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

CAL Syntax	Description	Octal Code
Ai PSj	Population count of (Sj) to Ai	026ij0
Ai QSj	Population count parity of (Sj) to Ai	026ij1

The 026ij0 instruction counts the number of bits set to 1 in (Sj) and enters the result into the low-order 7 bits of Ai. The high-order 17 bits of Ai are zeroed.

The 026ij1 instruction counts the number of bits set to 1 in (Sj). Then, the low-order bit, which shows the odd/even state of the result is transferred to the low-order bit position of the Ai register. The high-order 23 bits are cleared. The actual population count is not transferred.

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 case

(Ai) = 0 if j = 0

CAL Syntax	Description	Octal Code
Ai ZSj	Leading zero count of (Sj) to Ai	027ijx

This instruction counts the number of leading zeros in Sj and enters the result into the low-order 7 bits of Ai. The high-order 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

CAL Syntax		Description	Octal Code
Ai	Aj+Ak	Integer sum of (Aj) and (Ak) to Ai	030ijk
Ai	Ak§	Transmit (Ak) to Ai	030i0k
Ai	Aj+1§	Integer sum of (Aj) and 1 to Ai	030ij0
Ai	Aj-Ak	Integer difference (Aj) less (Ak) to Ai	031ijk
Ai	-1§	Transmit -1 to Ai	031i00
Ai	-Ak§	Transmit the negative of (Ak) to Ai	031i0k
Ai	Aj-1§	Integer difference (Aj) less 1 to Ai	031ij0

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 CAL syntax form

030-031

Special cases

For 030:

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

For 031:

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

CAL Syntax	Description	Octal Code
Ai Aj*Ak	Integer product of (Aj) and (Ak) to Ai	032ijk

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) = 0 if j = 0

(Ak) = 1 if k = 0

Thus

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

CAL Syntax		Description	Octal Code
Ai	CI	Channel number of highest priority interrupt request to Ai	033i0x
Ai	CA,Aj	Current address of channel (Aj) to Ai	033ij0
Ai	CE,Aj	Error flag of channel (Aj) to Ai	033ij1

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

The channel number of the highest priority interrupt request is entered into Ai when the j designator is 0. 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 0. The error flag for the channel is entered into the low-order bit of Ai when the k designator is 1. 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.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai reserved

Aj reserved

Execution time

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

(Ai) = I/O error flag of channel (Aj) if

(Aj) ≠ 0 and k = 1

(Ai) = 0 if (Aj) = 1

2 CPs must elapse after an 0012jx instruction issue before issuing an 033i0x instruction.

CAL Syntax	Description	Octal Code
Bjk,Ai ,A0	Block transfer (Ai) words from memory starting at address (A_0) to B registers starting at register jk	034ijk
Bjk,Ai 0,A0§	Block transfer (Ai) words from memory starting at address (A_0) to B registers starting at register jk	034ijk
,A0 Bjk,Ai	Block transfer (Ai) words from B registers starting at register jk to memory starting at address (A_0)	035ijk
0,A0 Bjk,Ai§	Block transfer (Ai) words from B registers starting at register jk to memory starting at address (A_0)	035ijk
Tjk,Ai ,A0	Block transfer (Ai) words from memory starting at address (A_0) to T registers starting at register jk	036ijk
Tjk,Ai 0,A0§	Block transfer (Ai) words from memory starting at address (A_0) to T registers starting at register jk	036ijk
,A0 Tjk,Ai	Block transfer (Ai) words from T registers starting at register jk to memory starting at address (A_0)	037ijk
0,A0 Tjk,Ai§	Block transfer (Ai) words from T registers starting at register jk to memory starting at address (A_0)	037ijk

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 low-order 7 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.

§ Special CAL syntax form

The first memory location referenced by the transfer instruction is specified by (A_0). The A_0 register contents are not altered by execution of the instruction. Memory references are incremented by 1 for successive transfers.

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

Hold issue conditions

A0 through A_7 reserved (034, 036)

A0 Ai, or S_0 through S_7 reserved (035, 037)

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

034 - 037 in process

Exchange in process

Scalar reference in CP 2

Rank B data valid

Fetch request in previous 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. (Ai) = 0 causes a zero-block transfer.

   (Ai) in the range greater than 100_8 and less than 200_8 causes a wrap-around condition.

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

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

CAL Syntax	Description	Octal Code
Si exp	Transmit jkm to Si	040ijkm
Si exp	Transmit complement of jkm to Si	04lijkm

These 2-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 high-order bits of 0.

The 04l instruction enters into Si a 64-bit value that is the complement of a value formed by the 22-bit jkm field and 42 high-order bits of 0. The complement is formed by changing all 1 bits to 0 and all 0 bits to 1. Thus, for the 04l instruction, the high-order 42 bits of Si are set to 1 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

CAL Syntax	Description	Octal Code
Si <exp	Form exp = 64-jk bits of ones mask in Si from right	042ijk
Si # >exp§	Form exp = jk bits of zeros mask in Si from left	042ijk
Si 1§	Enter 1 into Si	042i77
Si -1§	Enter -1 into Si	042i00
Si >exp	Form exp = jk bits of ones mask in Si from left	043ijk
Si # <exp§	Form exp = 64-jk bits of zeros mask in Si from left	043ijk
Si 0§	Clear Si	043i00

The 042 instruction generates a mask of 64-jk ones from right to left in Si. For example, if jk = 0, Si contains all 1 bits and if jk = 77_8, Si contains zeros in all but the low-order bit.

The 043 instruction generates a mask of jk ones from left to right in Si. For example, if jk = 0, Si contains all 0 bits and if jk = 77_8, Si contains ones in all but the low-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

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
Si Sj&Sk	Logical product of (Sj) and (Sk) to Si	044ijk
Si Sj&SB§	Sign bit of (Sj) to Si	044ij0
Si SB&Sj§	Sign bit of (Sj) to Si (j ≠ 0)	044ij0
Si #Sk&Sj	Logical product of (Sj) and complement of (Sk) to Si	045ijk
Si #SB&Sj§	(Sj) with sign bit cleared to Si	045ij0
Si Sj Sk	Logical difference of (Sj) and (Sk) to Si	046ijk
Si Sj SB§	Toggle sign bit of (Sj), then enter into Si	046ij0
Si SB Sj§	Toggle sign bit of (Sj); then enter into Si (j ≠ 0)	046ij0
Si #Sj Sk	Logical equivalence of (Sk) and (Sj) to Si	047ijk
Si #Sk§	Transmit ones complement of (Sk) to Si	047i0k
Si #Sj\SB§	Logical equivalence of (Sj) and sign bit to Si	047ij0
Si #SB\Sj§	Logical equivalence of (Sj) and sign bit to Si (j ≠ 0)	047ij0
Si #SB§	Enter ones complement of sign bit into Si	047i00
Si Sj!Si&Sk	Scalar merge	050ijk
Si Sj!Si&SB§	Scalar merge of (Si) and sign bit of (Sj) to Si	050ij0
Si Sj!Sk	Logical sum of (Sj) and (Sk) to Si	051ijk
Si Sk§	Transmit (Sk) to Si	051i0k
Si Sj!SB§	Logical sum of (Sj) and sign bit to Si	051ij0
Si SB!Sj§	Logical sum of (Sj) and sign bit to Si (j ≠ 0)	051ij0
Si SB§	Enter sign bit into Si	051i00

§ Special CAL syntax

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 1 when the corresponding bits of (Sj) and (Sk) are 1 as in the following example:

```
(Sj) = 1 1 0 0
(Sk) = 1 0 1 0
(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 0. The sign bit of (Sj) is extracted into Si if the j designator is non-zero and the k designator is 0.

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 1 when the corresponding bits of (Sj) and the complement of (Sk) are 1 as in the following example:

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

where (Sk') = complement of (Sk)

Si is cleared if the j and k designators have the same value or if the j designator is 0. (Sj) with the sign bit cleared is transmitted to Si if the j designator is non-zero and the k designator is 0.

The 046 instruction forms the logical difference (exclusive OR) of (Sj) and $\overline{(Sk)}$ and enters the result into Si. Bits of Si are set to 1 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 non-zero value. (Sk) is transmitted to Si if the j designator is 0 and the k designator is non-zero. The sign bit of (Sj) is complemented and the result is transmitted to Si if the j designator is non-zero and the k designator is 0.

The 047 instruction forms the logical equivalence of (Sj) and (Sk), and enters the result into Si. Bits of Si are set to 1 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 non-zero value. The complement of (Sk) is transmitted to Si if the j designator is 0 and the k designator is non-zero. All bits except the sign bit of (Sj) are complemented and the result is transmitted to Si if the j designator is non-zero and the k designator is 0. The result is the complement of that produced by the 046 instruction.

The 050 instruction merges the contents of (Sj) with (Si) depending on the ones mask in Sk. The result is defined by the following Boolean equation:

$$(Si) = (Sj)(Sk) + (Si)(Sk')$$

where Sk' is the complement of Sk as illustrated:

```
(Sk)  = 1 1 1 1 0 0 0 0
(Sk') = 0 0 0 0 1 1 1 1
(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 1 if the j designator is 0 and the k designator is non-zero. The sign bit of (Sj) replaces the sign bit of Si if the j designator is non-zero and the k designator is 0. The sign bit of Si is cleared if the j and k designators are both 0.

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 1 of the corresponding bits of (Sj) and (Sk) is set as in the following example:

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

044-051

(Sj) is transmitted to Si if the j and k designators have the same non-zero value. (Sk) is transmitted to Si if the j designator is 0 and the k designator is non-zero. (Sj) with the sign bit set to 1 is transmitted to Si if the j designator is non-zero and the k designator is 0. A ones mask consisting of only the sign bit is entered into Si if the j and k designators are both 0.

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si, Sj, and Sk reserved

Execution time

Instruction issue, 1 CP

Si ready, 1 CP

Special cases

(Sj) = 0 if j = 0

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

CAL Syntax	Description	Octal Code
S0 Si <exp	Shift (Si) left exp = jk places to S_0	052ijk
S0 Si >exp	Shift (Si) right exp = 64-jk places to S_0	053ijk
Si Si <exp	Shift (Si) left exp = jk places to Si	054ijk
Si Si >exp	Shift (Si) right exp = 64-jk places to Si	055ijk

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

Instruction issue, 1 CP

For 052, 053, S_0 ready, 2 CPs

For 054, 055, Si ready, 2 CPs

Special cases

None

CAL Syntax	Description	Octal Code
Si Si,Sj<Ak	Shift (Si) and (Sj) left by (Ak) places to Si	056ijk
Si Si,Sj<1§	Shift (Si) and (Sj) left one place to Si	056ij0
Si Si<Ak§	Shift (Si) left (Ak) places to Si	056i0k
Si Sj,Si>Ak	Shift (Sj) and (Si) right by (Ak) places to Si	057ijk
Si Sj,Si>1§	Shift (Sj) and (Si) right one place to Si	057ij0
Si Si>Ak§	Shift (Si) right (Ak) places to Si	057i0k

These instructions are executed in the scalar shift unit. They shift 128-bit values formed by logically joining two S registers. Shift counts are obtained from register Ak. A shift of one place occurs if the k designator is 0.

All shifts are end-off with zero fill if i ≠ j. The shift is circular 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, provided i ≠ j.

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 0.

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 0.

All shifts counts, (Ak), are considered positive. All 24 bits of (Ak) are used for the shift count.

§ Special CAL syntax form

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si, Sj, or Ak reserved

Execution time

Instruction issue, 1 CP

Si ready, 3 CPs

Special cases

(Sj) = 0 if j = 0

(Ak) = 1 if k = 0

Circular shift if i = j ≠ 0 and (Ak) less than 64.

CAL Syntax		Description	Octal Code
Si	Sj+Sk	Integer sum of (Sj) and (Sk) to Si	060ijk
Si	Sj-Sk	Integer difference of (Sj) and (Sk) to Si	061ijk
Si	-Sk§	Transmit negative of (Sk) to Si	061i0k

These instructions are executed in the scalar add unit.

The 060 instruction forms the integer sums 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

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

For 060:

(Si) = (Sk) 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

§ Special CAL syntax form

CAL Syntax		Description	Octal Code
Si	Sj+FSk	Floating sum of (Sj) and (Sk) to Si	062ijk
Si	+FSk§	Normalize (Sk) to Si	062i0k
Si	Sj-FSk	Floating difference of (Sj) and (Sk) to Si	063ijk
Si	-FSk§	Transmit normalized negative of (Sk) to Si	063i0k

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 not. Underflow and overflow conditions are described in section 4.

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

Instruction issue, 1 CP

Si ready, 6 CPs

§ Special CAL syntax form

062-063

Special cases

For 062:

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

For 063:

(Si) = -(Sk)	normalized if (Sk) exponent is valid, j = 0 and k ≠ 0. Sign of (Si) is opposite that of (Sk) if (Sk) ≠ 0.
(Si) = (Sj)	normalized if (Sj) exponent is valid, j ≠ 0 and k = 0

CAL Syntax		Description	Octal Code
Si	Sj*FSk	Floating-point product of (Sj) and (Sk) to Si	064ijk
Si	Sj*HSk	Half-precision rounded floating-point product of (Sj) and (Sk) to Si	065ijk
Si	Sj*RSk	Rounded floating-point product of (Sj) and (Sk) to Si	066ijk
Si	Sj*ISK	Reciprocal iteration; 2-(Sj)*(Sk) to Si	067ijk

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 not.

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 19 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 4 under Floating-Point Arithmetic.

In the evaluation C = 2-B * A, B must be a reciprocal of A of less than 47 significant bits. Otherwise C will be in error. The reciprocal produced by the reciprocal approximation instruction meets this criterion.

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

If both exponent fields are 0, an integer multiply is performed. Correct integer multiply results and produced if the following conditions are met:

1. Both operand sign bits are 0.
2. The sum of the zero bits to the right of the least significant one bit in the two operands is greater than or equal to 48.

In this case the integer result obtained is the high-order 48 bits of the 96-bit product of the two operands.

CAL Syntax	Description	Octal Code
Si /HSj	Floating-point reciprocal approximation of (Sj) to Si	070ijx

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 part 2, section 4 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 48 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

(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

CAL Syntax		Description	Octal Code
Si	Ak	Transmit (Ak) to Si with no sign extension	071i0k
Si	+Ak	Transmit (Ak) to Si with sign extension	071i1k
Si	+FAk	Transmit (Ak) to Si as unnormalized floating-point number	071i2k
Si	0.6	Transmit constant 0.75 x 2^{48} to Si	071i3k
Si	0.4	Transmit constant 0.5 to Si	071i4k
Si	1.	Transmit constant 1.0 to Si	071i5k
Si	2.	Transmit constant 2.0 to Si	071i6k
Si	4.	Transmit constant 4.0 to Si	071i7k

These instructions perform functions that depend on the value of the j designator. The functions are concerned with transmitting information from an A register to an S register and with generating frequently used floating-point constants.

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 0 to normalize. For this instruction, the exponent in bits 2^{62} through 2^{48} 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 twos complement of Ak is entered into Si as the magnitude of the coefficient and bit 2^{63} of Si is set for the sign of the coefficient.

A sequence of instructions which would be used to convert to floating-point format, an integer whose absolute value is less than 24 bits is:

```
(CAL)  A1   S1
       S1   +FA1
       S    +FS1     9 CPs required.
```

When the j designator is 3, the floating-point constant of 0.75×2^{48} is entered into Si (4006060000000000000_8). This constant is used to create floating-point numbers from integer numbers (positive and negative) whose absolute value is less than 47 bits. A sequence of instructions which would be used for conversion of an integer in S1 is:

```
(CAL)  S2  0.6
       S1  S2-S1
       S1  S2-FS1     11 CPs required.
```

When the j designator is 4, the floating-point constant 0.5 (= 0 40000 4000 0000 0000 0000_8) is entered into Si.

When the j designator is 5, the floating-point constant 1.0 (= 0 40001 4000 0000 0000 0000_8) is entered into Si.

When the j designator is 6, the floating-point constant 2.0 (= 0 40002 4000 0000 0000 0000_8) is entered into Si.

When the j designator is 7, the floating-point constant 4.0 (= 0 40003 4000 0000 0000 0000_8) is entered into Si.

Hold issue conditions

034 - 037 in process

Exchange in process

Si register access conflict

Si reserved

Ak reserved (all instructions)

Execution time

Instruction issue, 1 CP

Si ready, 2 CPs

071

Special cases

(Ak) = 1 if k = 0

(Si) = (Ak) if j = 0

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

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

(Si) = 0.6×2^{60} (octal) if j = 3

(Si) = 0.4×2^{0} (octal) if j = 4

(Si) = 0.4×2^{1} (octal) if j = 5

(Si) = 0.4×2^{2} (octal) if j = 6

(Si) = 0.4×2^{3} (octal) if j = 7

CAL Syntax	Description	Octal Code
Si RT	Transmit (RTC) to Si	072ixx
Si VM	Transmit (VM) to Si	073ixx
Si Tjk	Transmit (Tjk) to Si	074ijk
Tjk Si	Transmit (Si) to Tjk	075ijk

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 1 each clock period. The real-time clock can be set 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, VM busy (VL) + 6 CPs

003 in process, VM not available until 6 CPs after 003 issue

Execution time

Instruction issue, 1 CP

For 072 through 074, Si ready, 1 CP

For 075, Tjk ready, 1 CP

Special cases

None

CAL Syntax	Description	Octal Code
Si Vj,Ak	Transmit (Vj element (Ak)) to Si	076ijk
Vi,Ak Sj	Transmit (Sj) to Vi element (Ak)	077ijk
Vi,Ak 0§	Clear Vi element (Ak)	077i0k

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 6 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, 1 CP

Special cases

(Sj) = 0 if j = 0

(Ak) = 1 if k = 0

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
Ai exp,Ah	Read from ((Ah) + jkm) to Ai	10hijkm
Ai exp,0§	Read from (jkm) to Ai	100ijkm
Ai exp,§	Read from (jkm) to Ai	100ijkm
Ai ,Ah§	Read from (Ah) to Ai	10hi000
exp,Ah Ai	Store (Ai) to (Ah) + jkm	11hijkm
exp,0 Ai§	Store (Ai) to jkm	110ijkm
exp, Ai§	Store (Ai) to exp	110ijkm
,Ah Ai§	Store (Ai) to (Ah)	11hi000
Si exp,Ah	Read from ((Ah) + jkm) to Si	12hijkm
Si exp,0§	Read from (exp) to Si	120ijkm
Si exp,§	Read from (exp) to Si	120ijkm
Si ,Ah§	Read from (Ah) to Si	12hi000
exp,Ah Si	Store (Si) to (Ah) + jkm	13hijkm
exp,0 Si§	Store (Si) to exp	130ijkm
exp, Si§	Store (Si) to exp	130ijkm
,Ah Si§	Store (Si) to (Ah)	13hi000

These 2-parcel instructions transmit data between memory and an A register or an S register. The content of Ah (treated as a 22-bit signed integer) is added to the signed 22-bit 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 high-order 40 bits of the memory word are ignored. On a store from Ai into memory, the high-order 40 bits of the memory word are zeroed.

§ Special CAL syntax form

10h-13h

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 bank conflict and unit busy 3 CPs

Rank B bank conflict and unit busy 2 CPs

Rank C bank conflict and unit busy 1 CP

Storage hold continuation

Ah reserved

For 10h only, Ai register access conflict

For 10h and 11h only, Ai reserved

For 12h and 13h only, Si reserved

For 12h only, Si register access conflict

Fetch request in previous clock period

176 in process unit busy (VL) + 4 CPs

177 in process unit busy (VL) + 5 CPs

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

For 10h, 12h only:

Rank A conflict, 3 CPs delay before Ai or Si ready

Rank B conflict, 2 CPs delay before Ai or Si ready

Rank C conflict, 1 CP delay before Ai or Si ready

For 12h only:

Hold storage, 1 CP delay if 070 register access conflict occurs (when the result entering coincides with a reciprocal approximation result entering Si).

CAL Syntax	Description	Octal Code
Vi Sj&Vk	Logical products of (Sj) and (Vk elements) to Vi elements	140ijk
Vi Vj&Vk	Logical products of (Vj elements) and (Vk elements) to Vi elements	141ijk
Vi Sj!Vk	Logical sums of (Sj) and (Vk elements) to Vi elements	142ijk
Vi Vk§	Transmit (Vk) elements to Vi elements	142i0k
Vi Vj!Vk	Logical sums of (Vj elements) and (Vk elements) to Vi elements	143ijk
Vi Sj/Vk	Logical differences of (Sj) and (Vk elements) to Vi elements	144ijk
Vi Vj/Vk	Logical differences of (Vj elements) and (Vk elements to Vi elements	145ijk
Vi 0§	Clear Vi elements	145iii
Vi Sj!Vk&VM	If VM bit = 1, transmit (Sj) to Vi elements If VM bit = 0, transmit (Vk elements) to Vi elements	146ijk
Vi #VM&Vk§	Vector merge of (Vk) elements and 0 to Vi elements	146i0k
Vi Vj!Vk&VM	If VM bit = 1, transmit (Vj elements) to Vi elements If VM bit = 0, transmit (Vk elements) to Vi elements	147ijk

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 0 of the Vi, Vj, or Vk register and increment the element number by 1 for each operation performed. All results are delivered to Vi.

§ Special CAL syntax form

For instructions 140, 142, 144, and 146, a copy of the content of Sj is delivered to the functional unit where it is held as one of the operands until the completion of the operation. 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 1 when the corresponding bits of (Sj) or (Vj element) and (Vk element) are 1 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 1 when one of the corresponding bits of (Sj) or (Vj element) and (Vk element) is 1 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 of Vi. Bits of an element are set to 1 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 2^{63} of the mask corresponds to element 0 of a V register. Bit 2^{0} 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 1, the first operand is transmitted; if bit n of the mask is 0, 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
(V600) = 1
(V601) = 2
(V602) = 3
(V603) = 4
```

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

```
(V700) = 1
(V701) = -1
(V702) = -1
(V703) = 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
(V200) = 1      (V300) = -1
(V201) = 2      (V301) = -2
(V202) = 3      (V302) = -3
(V203) = 4      (V303) = -4
```

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

```
(V100) = -1
(V101) = 2
(V102) = 3
(V103) = -4
```

   The remaining elements of V1 are unaltered.

140-147

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) is less than or equal to 5

Vi ready, (VL) + 4 CPs if (VL) greater than 5

Vj or Vk ready, 5 CPs if (VL) is less than or equal to 5

Vj or Vk ready, (VL) CPs if (VL) greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 4 CPs

Special cases

(Sj) = 0 if j = 0

For 145 only, if i = j = k, (Vi) = 0.

CAL Syntax	Description	Octal Code
Vi Vj <Ak	Shift (Vj) elements left by (Ak) places to Vi elements	150ijk
Vi Vj <1§	Shift (Vj) elements left one place to Vi elements	150ij0
Vi Vj >Ak	Shift of (Vj) elements right by (Ak) places to Vi elements	151ijk
Vi Vj >1§	Shift (Vj) elements right one place to Vi elements	151ij0

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 (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. All shift counts (Ak) are considered positive. All 24 bits of Ak are used for the shift count.

Unlike the 052-055 shift instructions, these instructions get the shift count from Ak, rather than the jk fields.

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

§ Special CAL syntax form

150-151

Execution time

Instruction issue, 1 CP

Vi ready, 11 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 6 CPs if (VL) greater than 5

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 6 CPs

Special case

(Ak) = 1 if k = 0

CAL Syntax	Description	Octal Code
Vi Vj,Vj<Ak	Double shifts of (Vj elements) left (Ak) places to Vi elements	152ijk
Vi Vj,Vj<1§	Double shifts of (Vj elements) left one place to Vi elements	152ij0
Vi Vj,Vj>Ak	Double shifts of (Vj elements) right (Ak) places to Vi elements	153ijk
Vi Vj,Vj>1§	Double shifts of (Vj elements) right one place to Vi elements	153ij0

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 high-order bits or the low-order 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. The operation starts with element 0 of Vj being joined with element 1 and the resulting 128-bit quantity is then shifted left by the amount specified by (Ak). The 64 high-order bits remaining are transmitted to element 0 of Vi. Figure 6-7 illustrates this operation.

If (VL) is 1, element 0 is joined with 64 bits of 0 and only the one operation is performed. If (VL) is greater than 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 in figure 6-8.

If (VL) is 2, however, element 1 is joined with 64 bits of 0 and only two operations are performed. In general, the last element of Vj as determined by (VL) is joined with 64 bits of zeros. Figure 6-9 illustrates this operation.

§ Special CAL syntax form

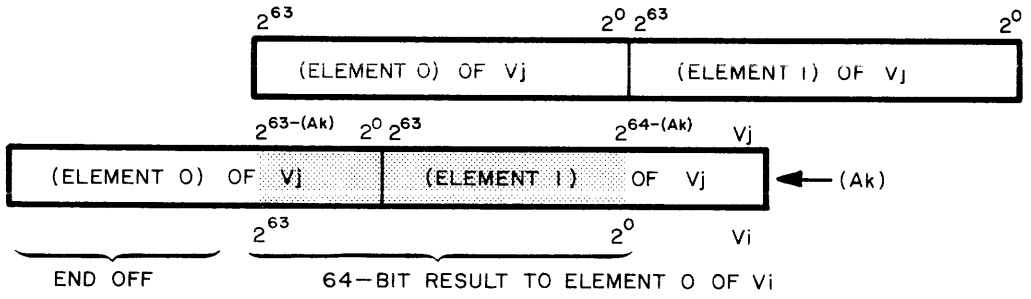


Figure 6-7. Vector left double shift, first element, VL greater than 1

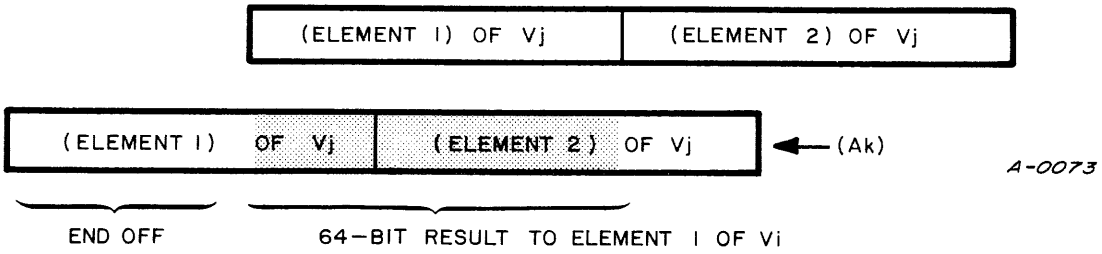


Figure 6-8. Vector left double shift, second element, VL greater than 2

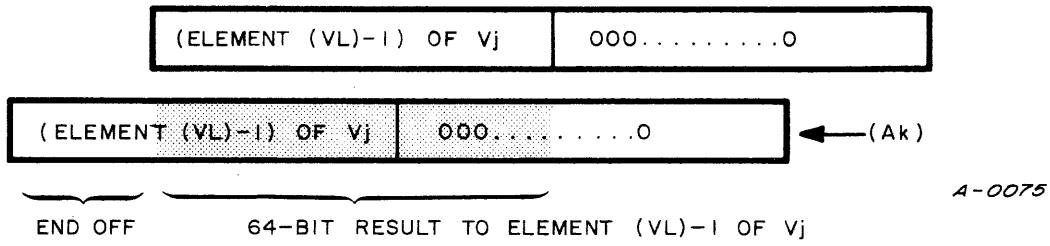


Figure 6-9. Vector left double shift, last element

If (Ak) is greater than 128, the result is all zeros. If (Ak) is greater than 64, the result register contains at least (Ak) - 64 zeros.

Example

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

```
(VL)   = 4
(Al)   = 3
(V400) = 0 00000 0000 0000 0000 0007
(V401) = 0 60000 0000 0000 0000 0005
(V402) = 1 00000 0000 0000 0000 0006
(V403) = 1 60000 0000 0000 0000 0007
```

Instruction 152541 is executed and following execution, the first four elements of V_5 contain the following values:

```
(V500) = 0 00000 0000 0000 0000 0073
(V501) = 0 00000 0000 0000 0000 0054
(V502) = 0 00000 0000 0000 0000 0067
(V503) = 0 00000 0000 0000 0000 0070
```

The 153 instruction performs right shifts. Element 0 of Vj is joined with 64 high-order bits of 0 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. Figure 6-10 illustrates this operation.

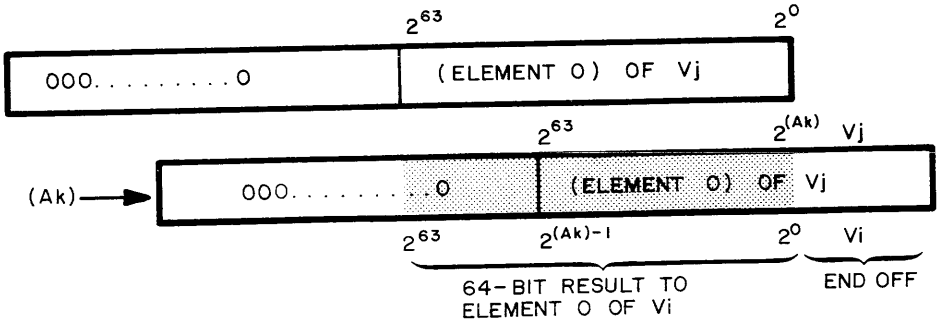


Figure 6-10. Vector right double shift, first element

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 in figure 6-11.

152-153

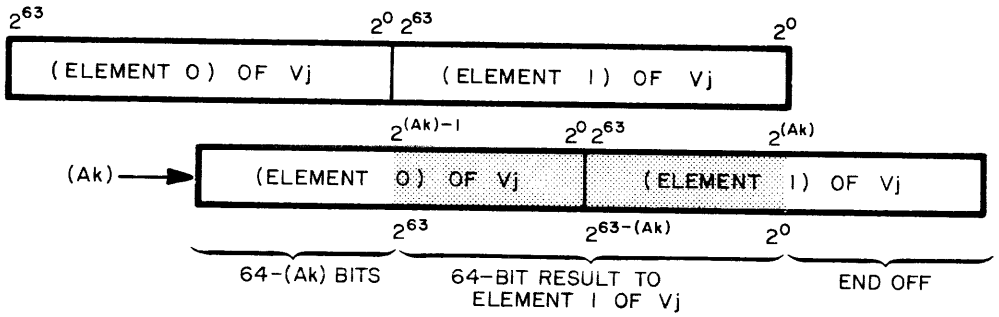


Figure 6-11. Vector right double shift, second element, VL greater than 1

The last operation performed by the instruction joins the last element of Vj as determined by (VL) with the preceding element. Figure 6-12 illustrates this operation.

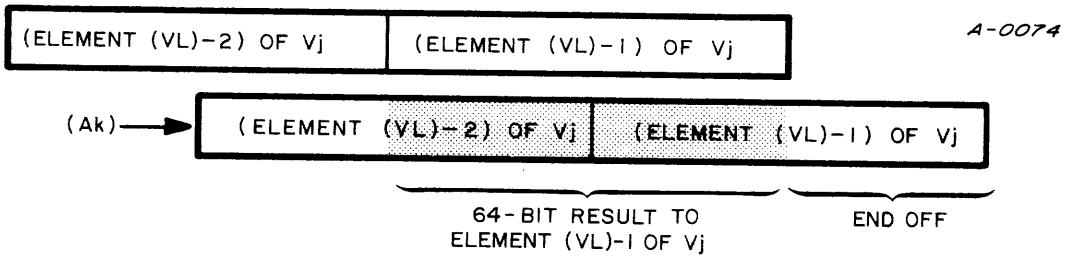


Figure 6-12. Vector right double shift, last operation

Example

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

```
(VL)   = 4
(A6)   = 3
(V200) = 0 00000 0000 0000 0000 0017
(V201) = 0 60000 0000 0000 0000 0006
(V202) = 1 00000 0000 0000 0000 0006
(V203) = 1 60000 0000 0000 0000 0007
```

Instruction 153026 is executed and following execution, register V0 contains the following values:

```
(V000) = 0 00000 0000 0000 0000 0001
(V001) = 1 66000 0000 0000 0000 0000
(V002) = 1 50000 0000 0000 0000 0000
(V003) = 1 56000 0000 0000 0000 0000
```

The remaining elements of V0 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) is less than or equal to 5

Vi ready, (VL) + 6 CPs if (VL) is greater than 5

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 6 CPs

Special case

(Ak) = 1 if k = 0

CAL Syntax		Description	Octal Code
Vi	Sj+Vk	Integer sums of (Sj) and (Vk elements) to Vi elements	154ijk
Vi	Vj+Vk	Integer sums of (Vj elements) and (Vk elements) to Vi elements	155ijk
Vi	Sj-Vk	Integer differences of (Sj) and (Vk elements) to Vi elements	156ijk
Vi	-Vk§	Transmit negative of (Vk elements) to Vi elements	156i0k
Vi	Vj-Vk	Integer differences of (Vj elements) and (Vk elements) to Vi elements	157ijk

These instructions are executed by the vector add unit.

Instructions 154 and 155 perform integer addition. Instructions 156 and 157 perform integer subtraction. The number of additions or subractions performed is determined by the contents of the VL register. All operations start with element 0 of the V registers and increment the element number by 1 for each operation performed. All results are delivered to elements of Vi. No overflow is detected.

Instructions 154 and 156 deliver a copy of (Sj) to the functional unit where the copy is retained as one of the operands until the vector operation completes. 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 156 and 157 only, Vj reserved

§ Special CAL syntax form

Execution time

Instruction issue, 1 CP

Vi ready, 10 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 5 CPs if (VL) is greater than 5

Vj or Vk ready, 5 CPs if (VL) is less than or equal to 5

Vj or Vk ready, (VL) CPs if (VL) is greater than 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)

CAL Syntax		Description	Octal Code
Vi	Sj*FVk	Floating-point products of (Sj) and (Vk elements) to Vi elements	160ijk
Vi	Vj*FVk	Floating-point products of (Vj elements) and (Vk elements) to Vi elements	161ijk
Vi	Sj*HVk	Half-precision rounded floating-point products of (Sj) and (Vk elements) to Vi elements	162ijk
Vi	Vj*HVk	Half-precision rounded floating-point products of (Vj elements) and (Vk elements) to Vi elements	163ijk
Vi	Sj*RVk	Rounded floating-point products of (Sj) and (Vk elements) to Vi elements	164ijk
Vi	Vj*RVk	Rounded floating-point products of (Vj elements) and (Vk elements) to Vi elements	165ijk
Vi	Sj*IVk	Reciprocal iterations; 2 - (Sj) * (Vk elements) to Vi elements	166ijk
Vi	Vj*IVk	Reciprocal iterations; 2 - (Vj elements) * (Vk elements) to Vi elements	167ijk

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 0 of the V registers and increment the element number by 1 for each successive operation.

Operands are assumed to be in floating-point format. Even-numbered instructions in the group deliver a copy of (Sj) to the functional unit where the copy is retained as one of the operands until the completion of the operation. 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 either operand is not normalized, there is no guarantee that the products will be normalized.

Out-of-range conditions are described in section 4.

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 result 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 19 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 19 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. See the description of the 067 instruction for more details.

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. See the description of the 067 instruction for more details.

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

160-167

Execution time

Instruction issue, 1 CP

Vi ready, 14 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 9 CPS if (VL) is greater than 5

Vj or Vk ready, 5 CPS if (VL) is less than or equal to 5

Vj or Vk ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 9 CPs

Special case

(Sj) = 0 if j = 0

CAL Syntax		Description	Octal Code
Vi	Sj+FVk	Floating-point sums of (Sj) and (Vk elements) to Vi element	170ijk
Vi	+FVk§	Transmit normalized (Vk elements) to Vi elements	170i0k
Vi	Vj+FVk	Floating-point sums of (Vj elements) and (Vk elements) to Vi elements	171ijk
Vi	Sj-FVk	Floating-point differences of (Sj) and (Vk elements) to Vi elements	172ijk
Vi	-FVk§	Transmit normalized negatives of (Vk elements) to Vi elements	172i0k
Vi	Vj-FVk	Floating-point differences of (Vj elements) and (Vk elements) to Vi elements	173ijk

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 0 of the V registers and increment the element number by 1 for each operation performed. All results are delivered to Vi. The results are normalized even if the operands are not.

Instructions 170 and 172 deliver a copy of (Sj) to the functional unit where it is retained as one of the operands until the completion of the operation. 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 4.

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

§ Special CAL syntax form

170-173

Execution time

Instruction issue, 1 CP

Vi ready, 13 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 8 CPs if (VL) is greater than 5

Vj and Vk ready, 5 CPs if (VL) is less than or equal to 5

Vj and Vk ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 8 CPs

Special case

(Sj) = 0 if j = 0

CAL Syntax	Description	Octal Code
Vi /HVj	Floating-point reciprocal approximation of (Vj elements) to Vi elements	174ij0

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 4 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 48 bits using the reciprocal iteration instruction.

Hold issue conditions

034 - 037 in process

Exchange in process

Vi or Vk reserved

174 in process, unit busy for (VL) + 4 CPs

Execution time

Instruction issue, 1 CP

Vi ready, 21 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 16 CPs if (VL) is greater than 5

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 16 CPs

Special case

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

CAL Syntax	Description	Octal Code
Vi PVj	Population count of (Vj elements) to Vi elements	174ij1
Vi QVj	Population count parity of (Vj elements) to Vi elements	174ij2

The 174ij1 instruction counts the number of bits set to 1 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 high-order bits of each Vi element are zeroed.

The 174ij2 instruction counts the number of bits set to 1 in each element of Vj. The least significant bit of each element result shows whether the result is an odd or an 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 remainder of the element is set to zeros. The actual population count results are not transferred.

These instructions use the vector population count unit, which shares some logic with the reciprocal approximation functional unit.

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) is less than or equal to 5

Vi ready, (VL) + 8 CPs if (VL) is greater than 5

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 8 CPs

CAL Syntax	Description	Octal Code
VM Vj,Z	VM = 1 where (Vj element) = 0	175xj0
VM Vj,N	VM = 1 where (Vj element) ≠ 0	175xj1
VM Vj,P	VM = 1 where (Vj element) positive, (bit 2^{63} = 0)	175xj2
VM Vj,M	VM = 1 where (Vj element) negative, (bit 2^{63} = 1)	175xj3

The 175xjk 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 2^{63} corresponds to element 0; bit 2^{0} corresponds to element 63.

The type of test made by the instruction depends on the low-order 2 bits of the k designator. The high-order bit of the k designator is not interpreted.

If the k designator is 0, the VM bit is set to 1 when (Vj element) is 0 and is set to 0 when (Vj element) is nonzero.

If the k designator is 1, the VM bit is set to 1 when (Vj element) is nonzero and is set to 0 when (Vj element) is zero.

If the k designator is 2, the VM bit is set to 1 when (Vj element) is positive and is set to 0 when (Vj element) is negative. A zero value is considered positive.

If the k designator is 3, the VM bit is set to 1 when (Vj element) is negative and is set to 0 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.

175

Hold issue conditions

034 - 037 in process

Exchange in process

Vj reserved

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

003 in process, VM busy 3 CPs

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

Execution time

Instruction issue, 1 CP

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

VM ready except for 073 instruction, (VL) + 4 CPs

VM 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

CAL Syntax	Description	Octal Code
Vi ,A0,Ak	Transmit (VL) words from memory to Vi elements starting at memory address (A_0) and incrementing by (Ak) for successive addresses	176ixk
Vi ,A0,1§	Transmit (VL) words from memory to Vi elements starting at memory address (A_0) and incrementing by 1 for successive addresses	176ix0
,A0,Ak Vj	Transmit (VL) words from Vj elements to memory starting at memory address (A_0) and incrementing by (Ak) for successive addresses	177xjk
,A0,1 Vj§	Transmit (VL) words from Vj elements to memory starting at memory address (A_0) and incrementing by 1 for successive addresses	177xj0

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 0 and are incremented by 1 for each transfer. Memory addresses begin with (A_0) and are incremented by the contents of Ak. Ak contains a signed 22-bit 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 two high-order bits of (Ak) are ignored.

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

§ Special CAL syntax form

176-177

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 (3 CPs maximum)

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 (assuming no bank conflicts):

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) is less than or equal to 5

Vi ready, (VL) + 9 CPs if (VL) is greater than 5

For 177 (assuming no bank conflicts):

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) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

Special cases

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

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

Inhibit I/O references

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

Special cases (continued)

(Ak) determines speed control. Successive addresses are located in successive banks. References to the same bank can be made every 4 CPs or more. Incrementing (Ak) by 16 (16-bank memory) or 8 (8-bank memory) places successive memory references in the same bank, so a word is transferred every 4 CPs. If the address is incremented by 8 (16-bank memory) or 4 (8-bank memory), 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, one word can transfer each CP.

CPU INTERFACES 7

INTRODUCTION

The CRAY-1 Computer is designed for use with front-end computers in a network of computers. Front-end interfaces connect the CRAY-1 I/O channels to channels of other computers. In S Series systems that do not have an I/O Subsystem, the interfaces are connected to CPU I/O channels. The interfaces adapt the CPU I/O channels to handle differences in voltage levels, grounding requirements, word sizes, data rates, and protocols. Each interface is designed to accommodate a specific type of computer, interfacing it to one CPU low-speed asynchronous I/O channel pair.

Standard interface hardware exists for the Maintenance Control Unit (MCU). Cray Research also offers standard interfaces for a variety of computers produced by other manufacturers and is continually expanding the list of front-end intefaces that it offers.

This section describes the physical construction, cabling limitations, and operation common to the interfaces. Specific information for each interface is contained in separate documentation.

PHYSICAL DESCRIPTION

Each interface is housed in a stand-alone cabinet (figure 7-1) located near the host computer. The cabinet is air-cooled, and operates directly from the 60 Hz AC power mains. The power consumption and therefore the heat generated by the interface cabinet varies with the complexity of the interface. The cabinet contains two or more logic modules and appropriate cabling connector panels. Internal power supplies provide the required logic and communication voltages. Cabinet grounding is flexible and can be configured to specific site requirements.

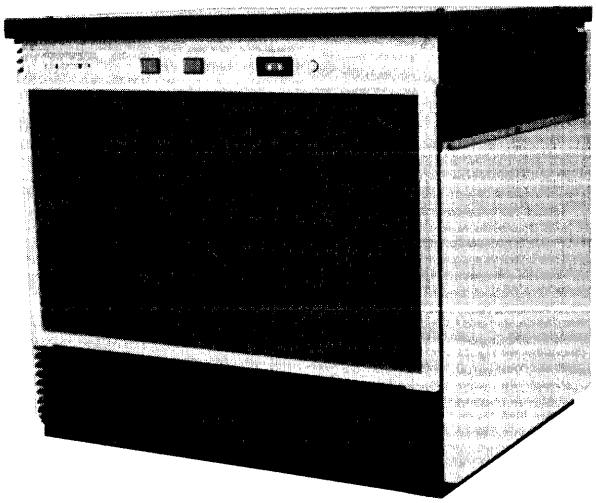


Figure 7-1. Typical interface cabinet

The MCU interface is built on a circuit board that is installed in the MCU computer chassis.

CABLING LIMITATIONS

An interface can be located as far as 320 cable feet from the CRAY-1 CPU. This generally allows enough distance to locate the interface cabinet near the front-end computer, easily meeting the cable length requirements of the front-end machine.

Connectors and cabling are supplied with the interface, so that it is ready to connect to host and CRAY-1 cabling.

The MCU interface has special cabling considerations.

OPERATION

The interface uses hardware logic to perform all command translation and protocol conversion needed to transfer data. Thus, its operation is invisible to the front-end user and the CRAY-1 user. The CRAY-1 is often treated simply as another peripheral device of the front-end system.

PART 3

I/O SUBSYSTEM

GENERAL INFORMATION 1

INTRODUCTION

The I/O Subsystem provides high-capacity data communications between the Central Processing Unit and peripheral devices, storage devices, and front-end computers. The I/O Subsystem is composed of two to four I/O Processors, a group of interfaces, and a Buffer Memory. This part of the manual describes these components.

The I/O Processor is a fast, multipurpose computer capable of extremely high data transfer rates. A 16-bit processor and a fast bipolar memory combine to support high-speed I/O operations. The input and output capabilities make the I/O Processor useful for network control, mass storage access, and computer interfacing.

Figure 1-1 shows the basic organization of an I/O Processor. The I/O Memory stores 65K parcels of 16 bits each. The control section has instruction stack, program exit stack, and control logic. The computation section consists of registers and functional units having interconnected data paths. The I/O section is based in six direct memory access (DMA) ports, and each port can handle data at approximately 850 Mbits per second. These DMA ports are expandable into multiple I/O channels for specialized interfacing needs.

MEMORY SECTION

The memory of an I/O Processor consists of four sections of four banks each of bipolar LSI (large scale integration) memory. All memory sections are independent of each other. The memory cycle time is 4 CPs meaning a section is ready for use again 4 CPs after the preceding memory reference. An exception to this is the I/O write operation which requires 6 CPs. The access time is the time required to bring an operand from memory to the accumulator, which is 7 CPs. Memory capacity is fixed at 65,536 parcels of 16 bits of data plus 2 odd parity bits for each parcel.

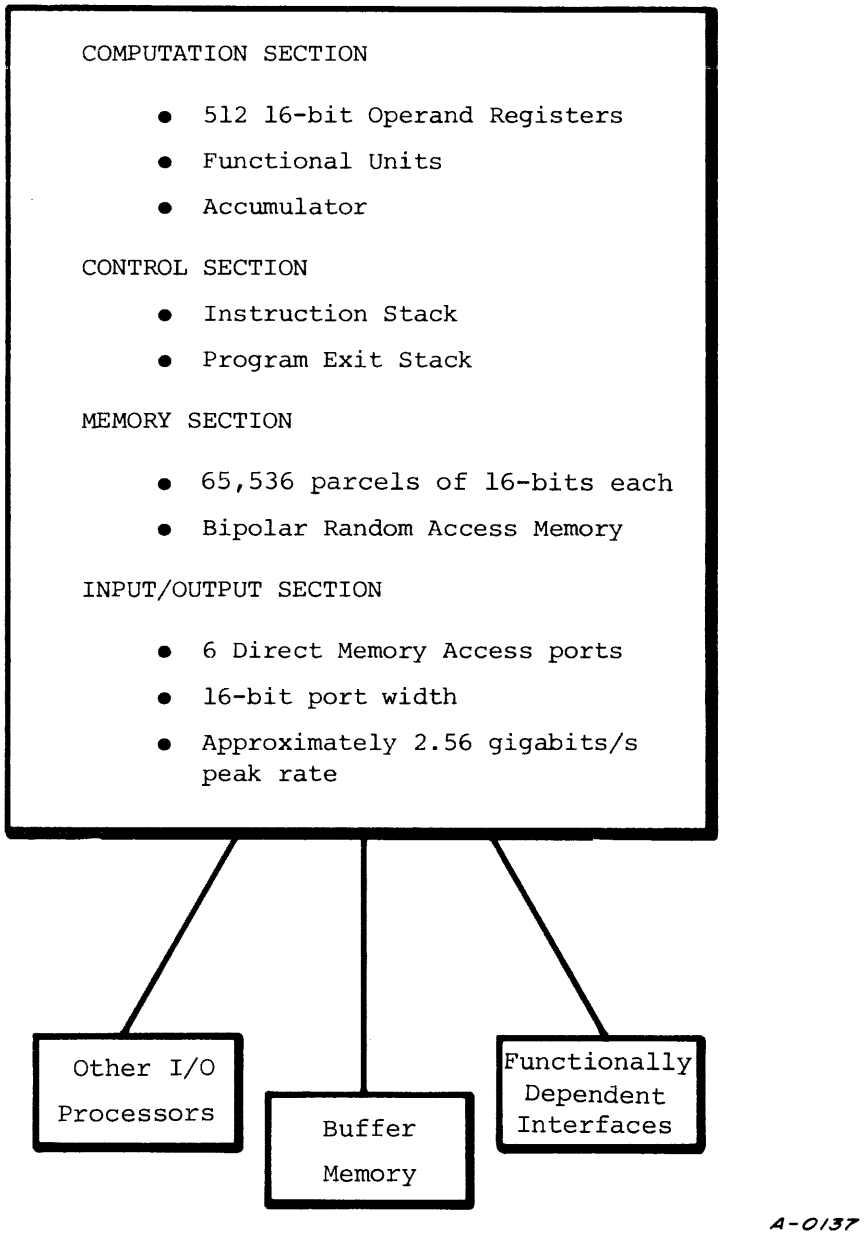


Figure 1-1. Basic organization of an I/O Processor

CONTROL SECTION

The I/O Processor executes 128 instruction codes as either 16-bit (1-parcel) or 32-bit (2-parcel) instructions. A wide assortment of branching and I/O instructions are included in the set. Instructions are stored in memory, and transferred into the instruction stack, under the control of a program address counter. Instructions issue from the instruction stack and are decoded into the control signals that enable the functions of the instruction.

The instruction stack provides fast access to a moving window of program instructions. The instruction stack holds 32 instruction parcels (16 bits each). The program is free to branch quickly about inside the stack, only slowing when more instructions must be read from memory. The instruction stack is actually a 32-parcel circular buffer: the parcels held for the longest time are overwritten by newly transferred instructions. Details are given in the control section discussion in this part of the manual.

The program exit stack is a last-in-first-out set of 16 registers that stores return addresses for program subroutine calls. The 16 registers provide for 14 nested levels of subroutines in the program; a pointer keeps track of the levels involved. An interrupt is generated when the stack is emptied or filled. The stack may be loaded or unloaded by the program. Therefore, the processor is not limited to the 14 nested levels, and it is possible to nest an unlimited number of subroutines by means of the software.

COMPUTATION SECTION

The computation section contains operating registers, functional units and an accumulator which operate together to execute a program of instructions stored in memory. All arithmetic (addition and subtraction) is in twos complement mode in a single adder. Floating-point arithmetic is not incorporated. A shift unit provides left or right shifts of up to 31 bit positions, either circular or end-off shifting. A logical product operation is provided.

Any of the 512 operand registers are used for temporary data storage or for indirect addressing to the memory. All operand registers are 16 bits wide.

The accumulator is a 16-bit signed register for temporary storage of operands or results. All movement of data within this single-address machine uses the accumulator either as the source of data or as the destination for results. The instruction code specifies the register or memory address from which data is brought to the accumulator or the register or memory address to which data is sent from the accumulator.

All transfers between memory and operand registers take place via the accumulator, and accumulator data can also be sent to or received from specialized I/O channels. A seventeenth accumulator bit occupies the 2^{16} bit position as the carry flag. When operations in the adder or shifter yield a carry, the accumulator 2^{16} bit is toggled. The carry flag is available for conditional testing.

INPUT/OUTPUT SECTION

Communication with the I/O Processor is through six direct memory access (DMA) ports. Each port is bidirectional and can transfer four 16-bit data words every 6 CPs. Input and output channels may be active at the same time, as long as they use different ports and are referencing different memory sections. The ports are assigned to channels with the possibility of several channels sharing one port. The slower the required data rate on the channels, the more channels may be multiplexed into one DMA port. The I/O Processor has a total of 40 channels.

Channels use Busy and Done flags for signalling the I/O Processor, and can communicate directly with the I/O Processor accumulator for control information. The use of the flags and the accumulator varies with the specific design of the channel interface logic. All channels communicate status and functions through the accumulator. Some low-speed devices may transfer data directly to and from the accumulator, using one of the channel registers.

I/O SUBSYSTEM CLOCK

The clock controlling the entire I/O Subsystem is a crystal-controlled oscillator running at 80 MHz. This gives a clock period of 12.5 nanoseconds. The speed can be adjusted slightly higher or lower, for maintenance purposes. When doing operations that require exact timing information, such as interval timing with the real-time clock, contact maintenance personnel to verify the clock period.

I/O MEMORY SECTION 2

INTRODUCTION

The I/O Processor memory, called the I/O Memory, consists of 16 banks of bipolar LSI (large scale integration) storage circuits. Each 4-bank section is independent of the other sections. The following paragraphs describe the memory speeds, memory organization, memory access, memory addressing, and memory parity protection.

Table 2-1. I/O Processor memory characteristics

- 65,536 parcels of 16 bits
- 16 banks of 4,096 parcels each
- 4 CP bank cycle time
- Read to accumulator in 7 CPs
- 1 instruction fetched/CP
- 1 data parcel/CP on sequential addressing
- Dual odd parity protection
- 6 full-duplex direct-memory-access ports

MEMORY SPEEDS

The memory cycle time is 4 CPs. The access time, that is, the time required to fetch an operand from memory to the accumulator, is 7 CPs. Instructions are fetched from memory at a rate of one parcel per clock period, in 4-parcel bursts. I/O operations transfer data in bursts of four parcels, one parcel each clock period. Due to the memory organization, explained later in this section, it is possible to reference sequential addresses every clock period, regardless of whether the operation is a read or a write. However, the same section can only be referenced once every 6 CPs. The data transferred in the operation may be one parcel of operand data or four parcels of I/O data.

MEMORY ORGANIZATION

The 16 banks of 4096 parcels each are divided into four sections of four banks each. Each section has separate access paths and sequence controls, as well as write data registers and read data registers. The four banks within a section share a common sequence control. A reference to any one bank of the section also references the other three banks in the section, but only the addressed bank transfers data.

A read reference to a memory section causes all four banks in that section to read out parcels to their read registers. The addressed parcel or parcels are then gated to the destination. In the case of an operand reference, a single parcel goes to the accumulator. In an I/O reference each of the four parcels is gated to the output channel in sequence. The read reference and transmission sequence are fixed with reference to timing of the initiation and moving of data. Once the reference has been started, it continues automatically. After 4 CPs, the section is then free to begin another overlapping reference.

A write sequence from the I/O section can be assembling data for a memory section in the bank write data registers at the same time that an I/O read reference is reading data from that same memory secton. The actual data moved per reference may vary from one operand parcel to a four parcel burst for I/O operation.

MEMORY ACCESS

Each memory section has three 16-bit data paths for reading and two 16-bit data paths for writing. One read path and one write path go to the accumulator. One read path and one write path go to the I/O section as Direct Memory Access (DMA) ports. The last read path goes to the instruction stack and carries instruction parcels. The DMA ports are explained in greater detail in the description of the I/O section later in this manual.

MEMORY ADDRESSING

A parcel in the 16-bank I/O Memory is addressed in 16 bits as shown in figure 2-1.

The low-order 4 bits specify one of the 16 banks. The next higher field specifies an address within the storage chip. The high-order bits select the storage chip.

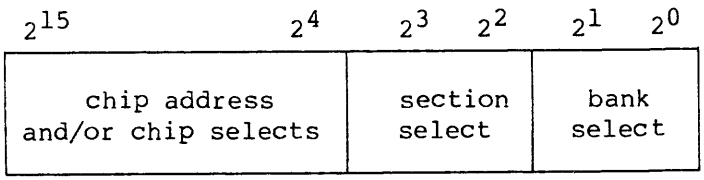


Figure 2-1. I/O Memory address format

Three address paths go to each of the four memory sections: one from the I/O, one from the accumulator, and one for instruction fetch references.

MEMORY PARITY PROTECTION

Data stored in memory is protected by an odd parity scheme. When a 16-bit word is stored in memory, 2 additional parity bits are generated and stored at the same address. One parity bit is assigned to the high-order 8 bits of the data word, and the other is assigned to the low-order 8 bits. Each parity bit is set or cleared to make the sum of 1 bit in the byte and parity bit an odd number. When the data is read from memory, it is checked for odd parity. If parity is even, an error has occurred in storage and an interrupt is generated.

The error handling routine may use the I/O Processor I/O Memory error channel to determine the location of the failing hardware and issue this information for maintenance people. Normally, no further I/O Processor operation would be attempted until the memory fault is corrected. The I/O Memory error channel is described with the Input/Output section in section 5.

IOP CONTROL SECTION 3

INTRODUCTION

The control section of an I/O Processor consists of an instruction control network, an instruction stack, and a program exit stack. Characteristics are summarized in table 3-1.

Table 3-1. Characteristics of IOP control section

- Single addressing mode
- 128 operation codes
- Instruction stack
- Program exit stack, 16 primary levels

Instructions can move data from a source to the accumulator or from the accumulator to a destination. Operand registers serve as temporary storage for operands and results. The functional units receive operand pairs and produce single results. In this single-address machine, one operand address is designated by the instruction and the other operand is contained in the accumulator. A typical data flow is: from memory to accumulator; from accumulator with an operand to functional unit and result back to accumulator; and from accumulator to memory. Figure 3-1 shows the organization of an I/O Processor and following paragraphs describe the elements of the control section.

INSTRUCTION CONTROL NETWORK

Figure 3-1 shows the general organization of the instruction control network for an I/O Processor. Important features are the instruction stack, II register, B register, RP and DP registers, exit stack, and P register. The decoding of operation codes into data path enablings, register reservations, and functional unit requirements are some of the activities of the instruction control network that are beyond the scope of this manual.

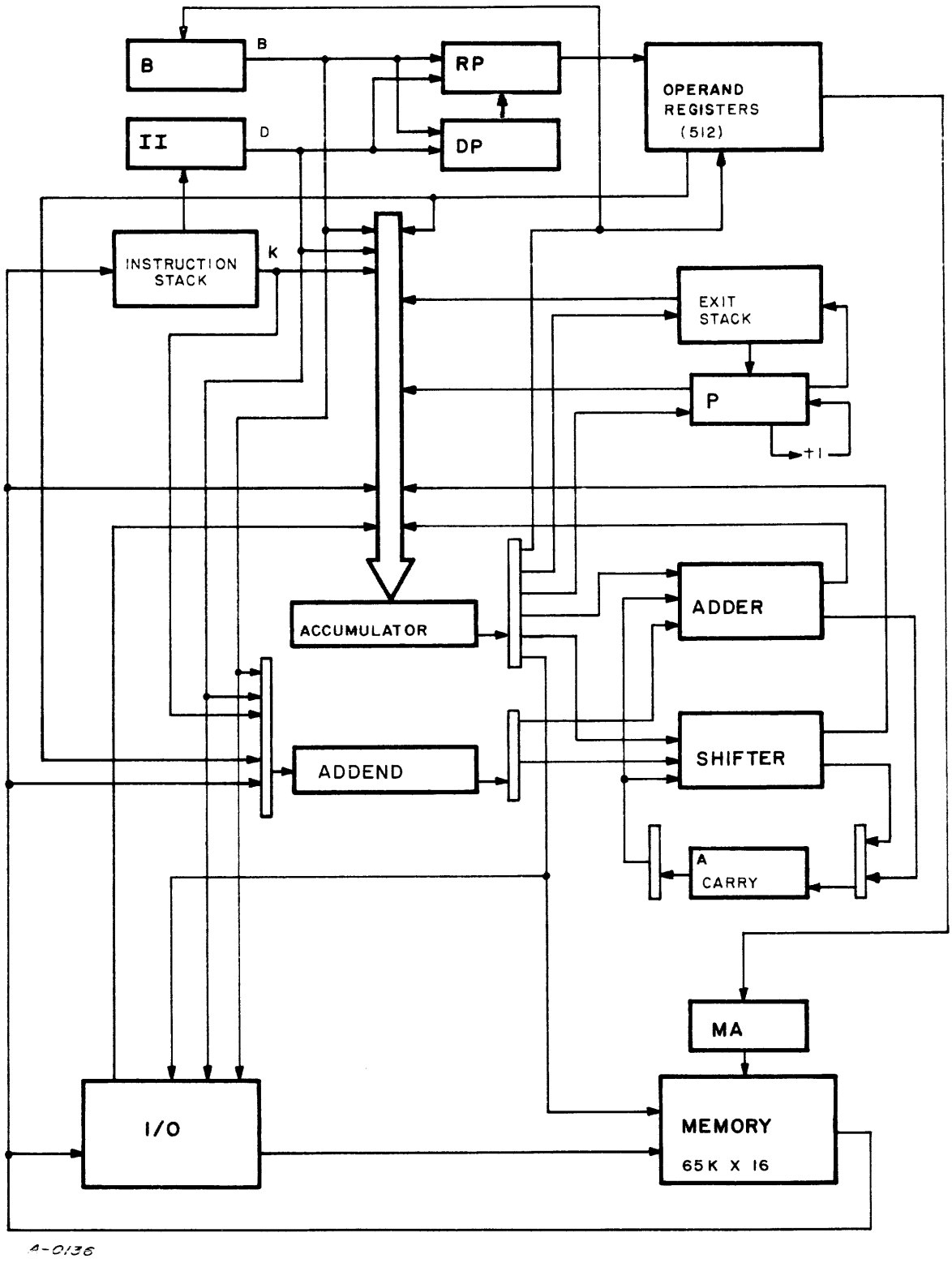


Figure 3-1. I/O Processor block diagram

INSTRUCTION STACK

Instructions occupy one or two parcels and consist of two designators and a constant field as shown in figure 3-2.

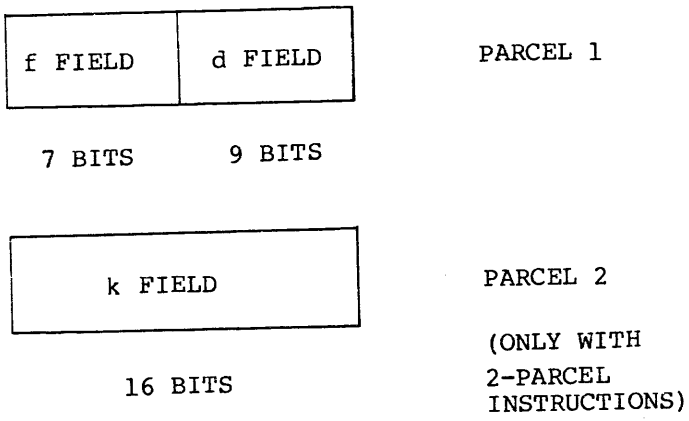


Figure 3-2. Instruction format

The 7-bit f designator is the operation code that specifies which instruction is to be executed, and also designates where and how the execution occurs. The 9-bit d field can contain data, address, or shift count. The 16-bit k field is a constant field that occupies the program parcel immediately following the d and f field parcel. A more detailed explanation of the instruction formats is given at the beginning of the Instructions section.

Program instructions are fetched from memory and stored in a 32-position, 16-bit instruction stack. Instructions are one parcel (16 bits) or two parcels (32 bits). The stack capacity for 32 instruction parcels allows short program loops to execute in the stack without reference to memory.

The instruction stack consists of two banks of registers with 16 parcels of program code in each bank (figure 3-3). The addresses alternate between the two banks so that the loading of data from storage can interleave with the readout of instructions for execution. Instructions are fetched from the memory in bursts of four parcels. Each burst references a storage address that is a multiple of four. The issue control selects which parcel of the four is received first. Loading is then continued sequentially. Memory supplies one parcel each CP if there are no memory conflicts.

The instruction passes directly into execution if the instruction sequence can issue in that clock period. The instruction is stored in the instruction stack regardless of whether it issues immediately or waits for issue. When an issue delay occurs, arriving parcels are stored in the instruction stack.

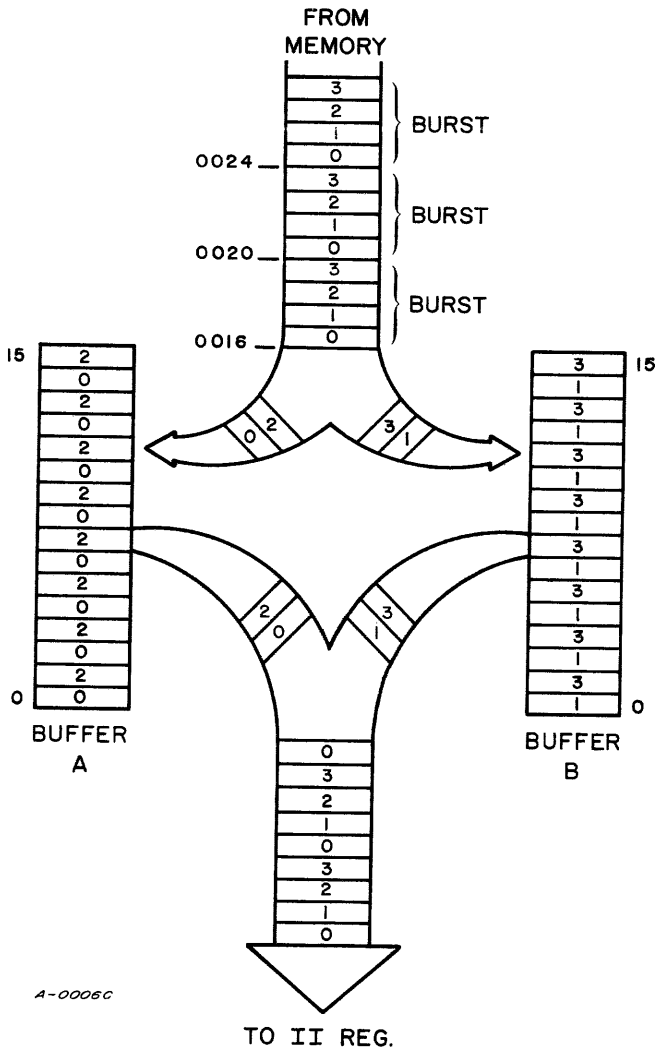


Figure 3-3. Instruction stack operation

The instruction stack is implemented as a circular buffer so that when the stack has been filled and new program code is required, the new code is stored in the beginning address of the stack. Loading continues circularly with the issue control circuits keeping a record of the current section of memory that is represented in the instruction stack. In an attempt to keep the next instructions to be executed in the stack, internal (background) fetches are performed as instructions execute sequentially.

In the following discussion, all numbers are octal.

Normally, two internal fetches are performed to keep the load point of the stack minimally 10_8 locations ahead of the next instruction to issue (stack point). Since fetches are performed on a 4-parcel boundary, the load pointer (L) can actually be 17_8 parcels ahead of the stack pointer (S) if S is at parcel 0 of a 4-parcel group. Once the condition $L-S \geq 10_8$ is satisfied, internal fetches are stopped. When S increments to the point that $L-S < 10_8$, then internal fetches resume. For sequential instructions, when the stack fills up, loading will begin again at location 0. Out-of-stack branch conditions will also cause a fetch beginning at stack location 0. Once this fetch is accomplished, barring other branches, the internal fetch mechanism will take over. All absolute branches (074-077, 120-137) are considered to be out-of-stack. Relative jumps (070-073, 100-117) may be in- or out-of-stack depending on the offset, d, and the location of the stack pointer.

Forward relative branch

Forward relative branches into the area of the stack which has been or will be satisfied by predetermined internal fetches will be in-stack, that is, a forward branch of 10_8 to 17_8 parcels. Normally, 13_8 is the maximum; but under certain conditions, a forward branch of up to 17_8 parcels is considered in-stack. In this case, an extra internal fetch is generated to retrieve the desired instructions. It is, however, impossible to predict when this will happen. Thus 10_8 parcels should be considered the upper limit for in-stack forward branches.

Backward relative branch

When the stack is filling for the first time after an out-of-stack condition, backward relative branches to stack location 0 are valid. After the stack is filling for succeeding times, backward branches into the area being loaded by internal fetches necessitates an out-of-stack condition. However, since the load pointer can only get a maximum of 17_8 parcels ahead of the stack pointer, a reverse jump of 23_8 parcels is guaranteed. If the stack pointer is at parcel 3 of a 4-parcel group, a reverse jump of 27_8 may be achieved. However, the maximum loop size guaranteed to be in-stack is 23_8 parcels.

II REGISTER

The II (Instruction Issue) register is a 16-bit register that receives the instruction parcel from the instruction stack. The instruction parcel may stay in the II register for more than 1 CP and leaves the II register when a new instruction is needed. The f field of the instruction parcel (2^9-2^{15}) is the operation code and is translated by logic associated with the II register to determine the particular sequence of operations required. Bits 2^0 - 2^8 of the instruction parcel are the d field, sent to the RP, DP or addend register, or to the accumulator. If the f field translation shows this parcel is the first of a 2-parcel instruction, the second parcel is sent from the instruction stack to the addend register or accumulator and is not interpreted as an instruction.

B REGISTER

The B register is a 9-bit address register used to designate one of the 512 operand registers. The B register is loaded from the accumulator, taking the low-order (2^0 - 2^8) bits. Accumulator bit 2^0 goes into B register 2^0 location. The contents of B may also address the I/O channel for an I/O instruction or may be used as an operand. In I/O instructions, the B register is an alternate for the instruction d field low-order bits 2^0 - 2^8 and can be altered by the program, in contrast to the d field which is part of the program.

RP REGISTER

The RP (Register Pointer) register is a 9-bit register that directly addresses one of the 512 operand registers for reading or writing. The RP register receives the d field from the II register on issue of each instruction using operand registers.

The operand registers are built to automatically read out data as addressed by the RP register, unless an instruction specifically demands a write into an operand register. The automatic read occurs each clock period, and if the read data is not needed, it is ignored.

DP REGISTER

The DP (Destination Pointer) register contains a 9-bit pointer that selects one of the 512 operand registers to receive the contents of the accumulator. The DP register receives the pointer from the D or B register bits 2^0-2^8. The pointer is sent when the instruction

issues. In a write operation from the accumulator to an operand register, there is usually a delay between the time the instruction issues and the time the register pointer is required. The DP register stores the pointer during the delay period.

A disadvantage exists if the next instruction calls for reading an operand register. Because the transfer of a pointer from the DP register to the RP register uses the same path into RP as is used by the pointer coming from II, instruction issue may be blocked until the path into RP is again free.

P REGISTER

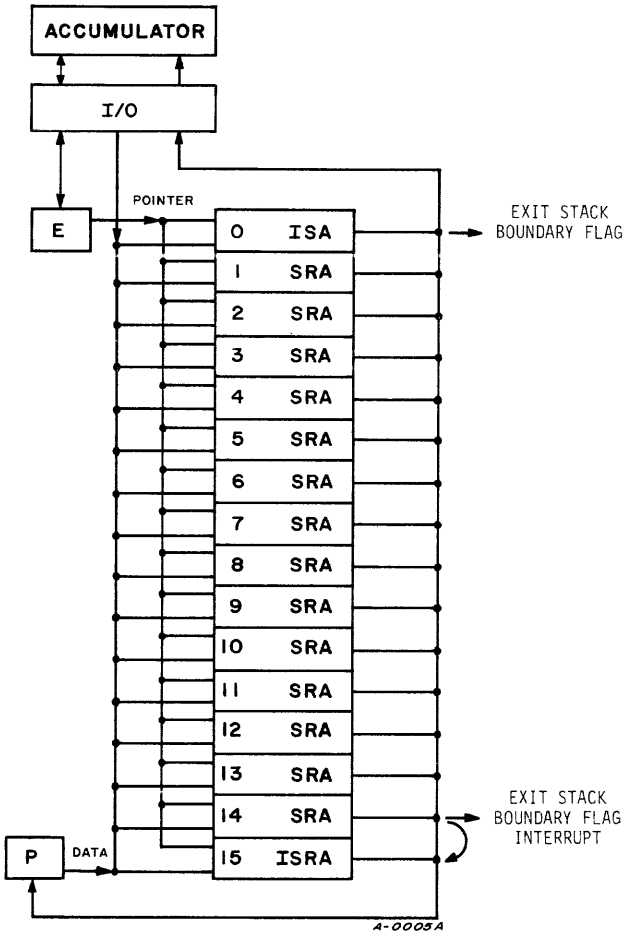
The 16-bit P (Program address) register holds the memory address of the instruction currently awaiting issue. The P register contents are automatically incremented as each instruction is executed in program sequence. A delay between reading from the instruction stack and issue keeps the address in P two program steps behind the instruction stack readout address. This delay is invisible to the programmer.

Branch instructions alter the P register content by either adding a positive or negative displacement value, or by entering an entirely new value.

PROGRAM EXIT STACK

The I/O Processor has a special hardware mechanism for storing the program return addresses when subroutines are called, and for storing the return addresses when the program is suspended to handle interrupts. This mechanism is the program exit stack, which consists of sixteen 16-bit registers. The program exit stack is addressed by a 4-bit pointer, the E register. The program can access and modify the contents of the program exit stack and the E register through I/O channel functions. The program exit stack is shown in figure 3-4.

The zero position in the stack is reserved for the interrupt handler starting address. This address is entered in the stack by the deadstart program and generally remains unaltered for the remainder of system execution. Interrupts cause the hardware to reference the stack zero position without reference to the E register.



ISA = Interrupt Start Address
SRA = Subroutine Return Address
ISRA = Interrupted Subroutine Return Jump Destination Address

Figure 3-4. Program exit stack

Positions 1 through 14 in the program exit stack are used for subroutine return addresses. A subroutine call from a routine advances the E pointer by 1 and then stores in the stack the return address at which the routine will continue when the called subroutine finishes. An exit from the subroutine to the routine reads the currently-pointed location in the stack for the return address. Then the E pointer decrements by one count to point to the higher level stack location. As a program goes into deeper levels of subroutines, the E count increases and more subroutine return addresses are stored. As the program exits out of subroutines, the E count drops back toward 0.

When the E pointer approaches the full limit of the stack, an exit stack interrupt is generated to allow the software to reconfigure the stack. The sequence is as follows:

1. E reaches 13
2. Subroutine call
3. E goes to 14
4. New return address loads to stack position 14
5. Interrupt sets
6. Return jump to new subroutine enters new value to P, but does not jump
7. Interrupt blocks further instruction issue (if system and channel interrupts are enabled)
8. Stack loads the address that was interrupted to position 15
9. Interrupt handler begins executing

When the E pointer reaches 0 and an exit instruction issues, an interrupt sets and the program jumps to the interrupt handler routine.

The sequence is as follows:

1. E reaches 0
2. Exit instruction occurs (normal or error)
3. Interrupt sets
4. Interrupt blocks further instruction issue (if system and channel interrupts are enabled)
5. Interrupt handler begins executing

If return jumps are used in an interrupt handler, it should be verified that there are enough levels left available in the stack. An interrupt with the exit stack pointer at 13_{10} will cause the pointer to go to 14_{10} and leave only one location open. A worse case exists if a return jump which causes a program fetch request (PFR) interrupt is issued with the stack pointer at 13_{10}. The return address will go in 14_{10} and the interrupt address with go into 15_{10}. This will now leave two interrupts present--both the exit stack boundary and PFR, with the PFR being the highest priority and no stack locations available. If the stack pointer is allowed to increment from 15_{10}, it will clear to 0, and incorrect return addresses will be used.

A suggested way of reconfiguring the program exit stack is to keep the stack half full. This gives wide freedom to call deeper levels or exit to higher levels. The deepest level interrupt is characterized by E = 15. When handling it, save the higher half (1-7) of the stack in memory,

move the lower half (8-15) to the higher part of the stack (1-8). This is the old 15 location that holds the interrupted subroutine return address. Exit back to the interrupted subroutine, and operation continues.

When a reconfigured stack exits to the E = 0 level, it exits directly to the interrupt handler routine. This highest level interrupt is characterized by E = 0 with the exit stack interrupt present. The handling routine can then rebuild the original higher half of the stack, reading it from memory. Set E to 7, and exit to that subroutine return address.

Program Exit Stack and I/O Interrupts

An I/O interrupt is treated much like a subroutine call. The interrupted program address is stored in the program exit stack at the next stack position and the entrance address for the interrupt routine is read from the zero position of the stack. On servicing the interrupt, the hardware clears the system interrupt enable flag to prevent other interrupts from interrupting the handling routine.

If return jumps are used in the interrupt handling routine, enough locations must be left in the exit stack to handle the maximum number of levels encountered in the interrupt handling routine. Then, when the interrupt handling routine is finished, it sets the system interrupt enable flag. The exit at the end of the handling routine reads from the exit stack the return address for the interrupted program.

Program Exit Stack Timing Note

After any modification to the stack locations or to the E pointer, at least 4 CPs must elapse before a program exit, return jump, or enabling system interrupts. A simple way to achieve the required delay is to read the modified contents back to the accumulator before exiting the routine. Modifying stack locations or the E pointer should only be done when system interrupts are disabled.

PROGRAM FETCH REQUEST FLAG

The program fetch request (PFR) flag is set during execution of jump instructions 074 - 077 and 120 - 137 when the instruction sequence finds a zero value in operand register d. This condition requires the monitor program to locate and fetch a segment of program code and to find a location in the I/O Memory for execution.

The setting of the program fetch request flag suspends execution of the current program sequence with an interrupt request at the completion of the interrupted instruction. The monitor program then reads the channel number through the channel 1 interface input register and interprets that number as the operand register requiring service.

MA REGISTER

The MA (Memory Address) register holds the address for an I/O Memory reference. The MA register contains 16 bits. It receives address information from an operating register, and holds it for I/O Memory use. This register is used for both read and write memory references.

IOP COMPUTATION SECTION 4

INTRODUCTION

The I/O Processor adds, subtracts, left shifts, and right shifts, using the adder, shifter and logical functional units. Temporary data storage is provided by a block of operand registers. All transfers to operand registers and all results from the functional units pass through the accumulator. These parts of the computation section are described in this section. Characteristics are summarized in table 4-1. Refer to the I/O Processor block diagram in the preceding section, which shows the organization the computation section in the I/O Processor.

Table 4-1. Characteristics of the I/O computation section

- 16-bit architecture
- Twos complement arithmetic
- Integer addition/subtraction unit
- Shift unit
- Logical operations
- 512 operand registers, 16 bits wide

OPERAND REGISTERS

Computation in the I/O Processor is supported by 512 operand registers. Each operand register contains 16 bits of data and has a 1 CP access time. The registers are addressed by the Register Pointer (RP) register. The only data path into the operand registers is from the accumulator. Operand register output data can go to either the accumulator or the addend register as operand data, or it can go to the Memory Address (MA) register as memory address data. The operand registers act as temporary locations for data, as index registers, and as indirect address registers for memory.

FUNCTIONAL UNITS

The computation section consists of an adder functional unit and a shifter functional unit. The computation section performs all the arithmetic required by the instruction set.

ADDER

The arithmetic logic of the I/O Processor is a twos complement adder that can also be used for subtraction. Adder operands come from the accumulator, instruction fields, B register, P register, operand registers, and memory. Except for the accumulator contents, operands come to the adder through the addend register. Adder results go to the accumulator for distribution, as needed and to the P register. The 17-bit operands are received from the accumulator and returned to the accumulator; the seventeenth bit corresponds to the carry bit of the accumulator. Operands from the addend register have 16 bits. The adder is also used by the branch instructions for P address calculations.

This adder can also be used for subtraction. In twos complement arithmetic, subtraction takes place by adding the ones complement of the subtrahend (the number subtracted from the minuend) to the minuend and then adding 1. When subtracting, the contents of the addend register are inverted and passed to the adder. The subtraction control signal is a 1, which is added to the intermediate result to give the final difference. Either an add or subtract occupies 1 CP; another clock period is required to put the results into the accumulator and carry bit register.

SHIFTER

The shifter implements the shifting instructions of the I/O Processor. It shifts up to 31 places left or right, either circularly or end-off with zero fill. The shifter receives the 17-bit accumulator data (including carry bit) to be shifted and the 5-bit addend register shift count. The shifted results, 17 bits, are returned to the accumulator and carry bit register. One CP is required for the shift; this is independent of the shift count and type of shift. Any shift instruction occupies a total of 2 CPs.

The maximum number of places an operand can be shifted is 31. If the shift count is 0 for a left or right shift, no shift occurs. With an end-off shift, if the count is greater than 16, the zero filling clears the result. In all shifts, the carry bit is treated as the highest order bit (2^{16}) of the operand and the result.

ACCUMULATOR

The accumulator is a 16-bit register that temporarily stores operands and results. Data from a wide variety of sources can be routed to the accumulator; many destinations for accumulator contents are available. Sources and destinations are listed in the table 4-2.

The accumulator is used to perform the logical product function. Logic at the input of the accumulator is enabled by the logical product instructions and creates the logical product of two operands. The result goes directly to the accumulator, with no extra time taken for the logical product function

Table 4-2. Accumulator sources and destinations

Sources	Destinations
B register II register d field II register k field Operand registers Adder/Shifter Memory I/O channels	Operand registers Adder/Shifter B register Memory I/O channels

Program branch instructions require arithmetic to form the destination address from two operands. These 070-137 instructions do not alter the content of the accumulator. Execution of these sequences is performed with a separate background accumulator not visible to the programmer.

CARRY-BIT REGISTER

The carry bit register is a 1-bit register that holds the carry generated in the adder or shifter. The carry bit is treated as if it were bit 2^{16} of the accumulator operand and is included in all adds, subtracts, and shifts.

The carry bit can be set by several conditional instructions that test I/O channel flags. The carry bit is also used as a criterion for many conditional jump and return jump instructions.

ADDEND REGISTERS

The addend register supplies operands to the adder and shifter. Whenever two operands are required, the accumulator supplies one operand and the addend register supplies the other. The 16-bit addend register receives data from either the B register, the instruction stack, the operating registers, or memory. Its only destinations are the adder and the shifter.

INPUT/OUTPUT SECTION 5

INTRODUCTION

The I/O Processor supports up to 40 channels for input or output use. There are six Direct Memory Access (DMA) ports to I/O memory, which can be used by these 40 channels. Some of the channels and one of the ports are assigned standard purposes for the system, but the remainder are free for peripheral device or Central Processing Unit support. This section describes I/O configuration, speeds, and channel characteristics as well as the interrupt scheme used. The standard channels are also covered in this section. Table 5-1 summarizes characteristics of the IOP input/output section.

Table 5-1. Characteristics of the IOP input/output section

- Supported by 6 full-duplex direct-memory-access ports
- Approximately 850 megabits/s per DMA port (maximum speed)
- 16 data bits, 2 status bits (Busy and Done)
- Channel number selected by instruction or register contents
- Simultaneous input and output via separate ports

I/O CONFIGURATION

The I/O channels are numbered octally; in general, an input channel of a channel pair is given an even number. The 12 standard channels (numbers $0\text{-}13_8$) are the same among all of the I/O Processors of the computer system. Channels 014_8 through 017_8 may be channel options or may be an optional group of channels. The 020_8 through 047_8 channels are implemented as optional groups of four channels. Channels 014_8 through 047_8 are variable among the I/O Processors, depending on the overall system configuration. Five DMA ports to the I/O memory are available to the interfaces associated with channels 014_8 through 047_8.

(The sixth DMA port connects to the Buffer Memory.) Faster devices connect to an I/O Processor via the DMA ports, which allow block transfers. The slower devices can be supported via the accumulator channels. Several devices may be interfaced to share a single DMA port among the devices while each device has a unique channel number. This method is used in supporting groups of four disk storage units and Block Mux Channels.

I/O SPEEDS

The DMA ports are each capable of transferring a block of data at the approximate rate of 850 Mbits per second. Only 3 DMA ports can be active at once. DMA ports may be transferring data into I/O Memory, while other DMA ports simultaneously are transferring data from I/O Memory. This gives a maximum memory data rate of approximately 2.56 billion bits a second.

The maximum speed of accumulator channels depends on the speed of the interrupt service routine.

CHANNEL CHARACTERISTICS

The operating characteristics for the accumulator channels and the channels using DMA ports are similar in many respects. The following descriptions outline the control and data signals that are used and give the requirements for each signal. The channels are described independently of the interfaces that may be connected to them.

ACCUMULATOR CHANNELS

Each accumulator channel uses the following signals:

Function designators
Function strobe
Accumulator data
Busy/done flag to carry bit
Read done
Read busy
Master clear
Clock
Interrupt

Within each interface are two 1-bit registers comprising the done and busy flags for the channel. The interface is able to set or clear these flags, and the I/O Processor can sample them by means of the read done and read busy control signals.

Function Designators

Bits 2^0 - 2^3 of the f field of I/O instructions are used as a function code ($0 - 17_8$) to an interface (see the Instructions section in this part of the manual). This function is interpreted by the interface in a manner unique to the interface. The same function code can mean entirely different operations to different interfaces. The function code is 4 bits sent on lines from the I/O Processor instruction logic to the interface, prior to any other channel action. The function code may be stable on the lines for only 1 CP.

Function Strobe

The function strobe signal accompanies the function code bits. This signal alerts the interface to the presence of the function code.

Accumulator Data

The use of the accumulator data depends on the function code and the particular interface. For example, it may be treated as a parameter for a Buffer Memory transfer, or as a character for a display. The outgoing data from the accumulator is reliable only for the clock period containing the function strobe signal. Data coming from the interface to the accumulator also depends on the function code and interface.

Read Done

When the read done signal is sent to the interface, the interface done flag is sent back to the accumulator carry bit. The done flag is carried on the busy/done signal line described below.

Read Busy

When the read busy signal is sent to the interface, the interface busy flag is sent back to the accumulator carry bit. The busy flag is carried on the busy/done signal line described below.

Busy/Done

The busy signal is a response to the read busy or read done signals to the interface. If the read busy signal has been received, the interface busy flag is copied to the busy/done line. If the done flag has been requested, the interface done flag is sent on the busy/done line. The busy/done flag simply carries the set or cleared state of the requested flag. The signal arrives back at the I/O Processor to enter the carry bit position. It becomes the new carry bit in the same clock period that the read busy/done instruction issues.

Master Clear

A master clear signal is sent to each interface when the I/O Processor is deadstarted. After that, the only master clear function would have to be an interface interpretation of one of the function codes as a master clear command.

Clock

The I/O Processor clock signal is sent to each interface to synchronize the logical operations. The clock signal is a pulse approximately 12.5 nanoseconds wide. The clock speed can be varied for maintenance, in which case the pulse width stays constant. This clock is similar to the clock used in the Central Processor; however, the two clock generators are independent.

Interrupt

The interrupt signal from an interface to the I/O Processor causes an interrupt request in the I/O Processor for the channel. An interrupt occurs if all of the following are true: (1) the interrupt condition remains present, (2) the interrupt enable flag is set for the interface channel, and (3) the system interrupt enable is set.

CHANNELS USING A DMA PORT

A channel using a DMA port is an accumulator channel with connections to the I/O Memory. In addition to the accumulator channel signals already described, the DMA channel also uses the following signals which are described next:

I/O Memory data
I/O Memory address
Request read
Request write
Acknowledge write

I/O Memory Data

A group of 16 lines carries data from I/O Memory to the interface. A second group of 16 lines carries interface data to the I/O Memory. Data is transferred in groups of four 16-bit parcels, one parcel per clock period in sequential clock periods. When writing data into I/O Memory, the first parcel must be sent in the clock period after the acknowledge write is received from the I/O Processor. When reading data from I/O Memory, the data is received 7 CPs after the acknowledge read signal. In either case, the data is reliable for only 1 CP.

I/O Memory Address

The I/O Memory address comes from the interface to the I/O Processor on 14 lines. The interface receives the address from the accumulator under a specific function code, or the interface may be intelligent enough to create its own address. Any transfer of four parcels must have an address, which is the I/O Memory address for the first parcel of the 4-parcel group. The I/O Processor logic then takes care of incrementing the address for the later three parcels of the group. The I/O Memory address must be stable during the clock period of the request read or the request write signals.

Request Read

The request read signal is sent from the interface to the I/O Processor to take data from I/O Memory. This signal accompanies the I/O Memory address bits to the I/O Processor.

Request Write

The request write signal is sent from the interface to the I/O Processor when data is to be sent to I/O Memory. This signal accompanies the I/O Memory address bits to the I/O Processor.

Acknowledge Read

The acknowledge read signal is sent from the I/O Processor when the read requested by the interface can be performed. It has a minimum delay after the request read of 1 CP, but the delay may be stretched by memory conflicts. It occurs 7 CPs before the first parcel of data of the 4-parcel group and lasts 1 CP.

Acknowledge Write

The acknowledge write signal is sent from the I/O Processor when the requested write can be performed. It has a minimum delay after the request write signal of 1 CP, but memory conflicts may increase the delay. The interface must place the first parcel of data on the lines to the I/O Processor in the clock period after the acknowledge write signal is received at the interface.

READ SEQUENCE

The interface begins the read from I/O Memory after being issued an I/O instruction that commands the interface to start a transfer from I/O Memory to a peripheral. An address may be in the accumulator when the instruction issues and the interface may take that address as the starting address. Then, the interface sends a request read signal and the address bits to the I/O Processor. After a minimum delay of 1 CP, the acknowledge read signal is received at the interface. After 7 CPs, the interface takes the first parcel. It then takes the three following parcels in 3 CPs. To read another four parcels from I/O Memory to the interface, another request read and address must be sent from the interface.

WRITE SEQUENCE

The interface begins the write into I/O Memory after being issued an I/O instruction that commands the interface to start a transfer from a peripheral to I/O Memory. The address for the transfer may be contained in the accumulator at the time of instruction issue, or other means may be used to generate the address. Then, the interface sends a request write signal and the address bits to the I/O Processor. After a minimum delay of 1 CP, the acknowledge write signal is received from the I/O Processor. The interface must send the first parcel of data to the I/O Processor in the clock period after the acknowledge write signal was received. Each of the next 3 CPs transfers another parcel into the I/O Memory.

If another group of four parcels is to be written into I/O Memory, another request write and address must be sent from the interface.

STANDARD CHANNELS

Standard functions are assigned to 12 of the 40 available channels, which are the same for all I/O Processors. The interface logic has been built into each I/O Processor to handle each standard channel. The functions, channel numbers, and A Programming Machine Language (APML) mnemonics are listed in table 5-2. Another six standard channels interconnect the system I/O Processors as shown in the table. The standard channels are all accumulator channels except for the one DMA port used by the Buffer Memory.

Table 5-2. I/O Processor standard channel assignments

	Channel Number	Mnemonic		Function
All IOPs	000	IOR		Interrupt request
	001	PFR		Program fetch request
	002	PXS		Program exit stack
	003	LME		I/O Memory error
	004	RTC		Real-time clock
	005	MOS		Buffer Memory interface
MIOP 0	006	AIA	BIOP	I/O Processor input
	007	AOA		I/O Processor output
	010	AIB	DIOP	I/O Processor input
	011	AOB		I/O Processor output
	012	AIC	XIOP or 2nd DIOP	I/O Processor input
	013	AOC		I/O Processor output

Table 5-3 lists the functions for each of the standard channels. These functions are explained in subsequent paragraphs.

Table 5-3. Standard channel functions

Device	Mnemonic	Function
I/O REQUEST CH. 0	IOR : 10	Read interrupt channel number
PROGRAM FETCH REQUEST CH. 1	PFR : 0	Clear the program fetch request flag
	PFR : 6	Clear the channel interrupt enable flag
	PFR : 7	Set the channel interrupt enable flag
	PFR : 10	Read the operand register number
PROGRAM EXIT STACK CH. 2	PXS : 0	Clear the exit stack boundary flag
	PXS : 6	Clear the channel interrupt enable flag
	PXS : 7	Set the channel interrupt enable flag
	PXS : 10	Read exit stack pointer, E
	PXS : 11	Read exit stack address, (E)
	PXS : 14	Enter exit stack pointer, E
	PXS : 15	Enter exit stack address, (E)
I/O MEMORY ERROR CH. 3	LME : 0	Clear the I/O Memory parity error flag
	LME : 6	Clear the channel interrupt enable flag
	LME : 7	Set the channel interrupt enable flag
	LME : 10	Read error information
REAL-TIME CLOCK CH. 4	RTC : 0	Clear the channel done flag
	RTC : 6	Clear the channel interrupt enable flag
	RTC : 7	Set the channel interrupt enable flag
	RTC : 10	Read real-time clock
BUFFER MEMORY CH. 5	MOS : 0	Clear the channel busy and done flags
	MOS : 1	Enter the I/O Memory address for next transfer
	MOS : 2	Enter upper portion of Buffer Memory address
	MOS : 3	Enter lower portion of Buffer Memory address
	MOS : 4	Read Buffer Memory to I/O Memory
	MOS : 5	Write Buffer Memory to I/O Memory
	MOS : 6	Clear the channel interrupt enable flag
	MOS : 7	Set the channel enable interrupt flag
	MOS : 14	Set the control flags
I/O PROCESSOR INPUT (AIA-AIC) CH. 6, 10, 12	AIA : 0	Clear the channel done flag
	AIA : 6	Clear the channel interrupt enable flag
	AIA : 7	Set the channel interrupt enable flag
	AIA : 10	Read input to accumulator and resume channel
I/O PROCESSOR OUTPUT (AOA-AOC) CH 7, 11, 13	AOA : 0	Clear the channel busy and done flags
	AOA : 1	Enter control bits from accumulator
	AOA : 6	Clear the channel interrupt enable flag
	AOA : 7	Set the channel interrupt enable flag
	AOA : 14	Set the channel busy flag and output accumulator data

CHANNEL FOR I/O REQUESTS (CH. 0)

The I/O Processor has the 0 channel number reserved for reading the interrupt requests. The only function implemented is the following.

IOR: 10 Read interrupt channel number

This function replaces the accumulator content with the highest priority channel number currently requesting an interrupt. The channel number is loaded into the low-order 9 bits of the accumulator content. The high-order bits are forced to 0. Only the four least significant bits are used for the channel number. A zero value means there are no unhandled channel interrupts.

The interface register used by this channel contains the number of the highest priority channel on which an interrupt is present. The value is changed either by clearing the interrupt enable flag for the appropriate channel or by clearing the done flag for that channel.

For this channel the done flag is always set and the busy flag is always cleared. This channel can be used with the 040-043 instructions to set or clear the carry flag.

CHANNEL FOR PROGRAM FETCH REQUEST (CH. 1)

The I/O Processor has an I/O channel that responds to the program fetch request flag when that flag is set. (The program fetch request flag is explained in the IOP control section of this manual.) This channel provides a mechanism for calling the I/O Processor monitor program when a new section of program code is required.

A 9-bit interface register holds the operand register number associated with the interrupt request. This register is cleared and a new number entered at the time the program fetch request flag is set.

PFR : 0§ Clear the program fetch request flag. This flag is treated as the channel done flag. There is no busy flag for this channel.

PFR : 6§§ Clear the channel interrupt enable flag. The program fetch request flag is not altered in this process.

PFR : 7§§ Set the channel interrupt enable flag. The program fetch request flag is not altered in this process.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR:10).

PFR : 10 Replace the content of the accumulator with the content of the interface register. The interface register content is loaded into the low-order 9-bit positions in the accumulator. The high-order bits are forced to 0. The content of the interface register remains unchanged until a new PFR occurs. The done flag is cleared.

CHANNEL TO PROGRAM EXIT STACK (CH. 2)

The I/O Processor has an I/O channel connected to the program exit stack hardware. This channel provides the monitor program with the information needed to reorganize the content of the program exit stack when the stack overflows.

PXS : 0§ Clear the exit stack boundary flag. This flag is treated as the channel done flag. There is no busy flag for this channel.

PXS : 6§§ Clear the channel interrupt enable flag. The exit stack boundary flag is not altered in this process.

PXS : 7§§ Set the channel interrupt enable flag. The exit stack boundary flag is not altered in this process.

PXS : 10 Load the E designator into the low-order 4 bits of the accumulator. The high-order bits of the accumulator are forced to 0.

PXS : 11 Replace the accumulator content with the content of the program exit stack address currently pointed by the E designator.

PXS : 14 Replace the E designator with the low-order 4 bits of the accumulator content.

PXS : 15 Enter the program exit stack with the accumulator contents at the address currently pointed by the E designator.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR:10).

CAUTION

The exit stack is both an I/O device and an integral part of the processor. Return and exit instructions and interrupts use the exit stack values immediately. The I/O channel access to the exit stack takes 4 CPs to complete. Time must be allowed for the I/O channel to complete the transfer before using the exit stack value by a return, exit or interrupt.

Any attempt to use a value changed in the stack within 5 CPs after it was changed has potentially disastrous effects on the program execution sequence.

A simple way to provide delay is to change the exit stack, then read the value back from the exit stack to the accumulator. Then do the exit or return.

For example:

NOT RECOMMENDED	RECOMMENDED
(E) = AAAB	(E) = AAAB
EXIT	A = (E) (delay)
	EXIT
or: A = AAAB	A = AAAB
PXS : 15	PXS : 15
EXIT	PXS : 11
	EXIT

Deadstart Sequence

On master clear, stack location 0 is cleared to a zero value. On the deadstart interrupt, P is therefore set to 0 and program execution begins at memory location 0. The exit stack pointer, E, is set to the current value plus 1 (wrapping around to 0 if the current value is 15.

CHANNEL FOR I/O MEMORY ERROR (CH. 3)

The I/O Processor has an I/O channel connected to the error detection circuits in the I/O Memory. This channel provides the error indication in the event of memory malfunction and provides maintenance information. The intent of this channel is to provide information helpful to the maintenance function as quickly as possible. No attempt at continued operation is expected beyond system shutdown.

LME : 0 Clear the I/O Memory parity error flag.

LME : 6§ Clear the channel interrupt enable flag.

LME : 7§ Clear the channel interrupt enable flag.

LME : 10 Read error information into the five lowest-order positions of the accumulator as follows.

Bit	Meaning
2^0	Bank number 2^0
2^1	Bank number 2^1
2^2	Section number 2^0
2^3	Section number 2^1
2^4	Error bit: 0 = Error in 2^0-2^7 byte 1 = Error in 2^8-2^{15} byte

CHANNEL TO REAL-TIME CLOCK (CH. 4)

The I/O Processor real-time clock (RTC) is a 17-bit counter/timer which interrupts the I/O Processor at 1-millisecond intervals. The real-time clock increments every clock period. Upon reaching a count of 234177_8, it sets the RTC channel done flag, clears to 0, and continues incrementing. There is no busy flag for this channel. Since the accumulator only holds 16 bits, the low-order bit of the counter is ignored, giving a timing accuracy of 2 CPs, and a count of 116077_8.

§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

There is no capability to set the real-time clock. To time an interval, the RTC must be read at the beginning and end of an interval. Thus to time an interval, the program must:

- Read the clock at the beginning of the interval,
- Store the value in a register and
- Read the clock at the end of the interval.

This instruction timing adds 8 CPs (4 clock counts) to the measured sequence. The algorithm for determining an interval without RTC interrupts is:

$$\text{Time (ns)}_8 = (\text{RTC ending}_8 - \text{RTC beginning}_8 - 4) \times 2 \text{ (CP ns)}$$

If RTC interrupts do occur during the interval, the algorithm becomes:

$$\text{Time (ns)}_8 = ((\text{RTC ending}_8 - \text{RTC beginning}_8 - 4) + (116077_8 \times \text{number of interrupts})) \times 2 \text{ (CP ns)}$$

For time intervals expected to be less than 1 millisecond, synchronize their beginnings with the occurrence of an RTC interrupt, thus eliminating the possibility of an RTC interrupt occurring sometime during the timing sequence.

The commands for the real-time clock channel are as follows:

RTC : $0^{\S}$	Clear the channel done flag.
RTC : $6^{\S\S}$	Clear the channel interrupt enable flag.
RTC : $7^{\S\S}$	Set the channel interrupt enable flag.
RTC : 10	Read the real-time clock count into the accumulator.

CHANNEL TO BUFFER MEMORY (CH 5.)

The I/O Processor has an I/O channel connected to the Buffer Memory. This channel allows the I/O processor program to transfer blocks of data in either direction between its I/O Memory and the Buffer Memory. The function requests for this channel are summarized below.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

This channel has three interface registers used to control the block copy operations. The Buffer Memory address is held in a 24-bit register. The high-order 15 bits of this address are entered with a function 2 request. The low-order 9 bits are entered with a function 3 request. The I/O Memory address is held in a 14-bit register which is entered with a function 1 request. The block length in Buffer Memory words is held in a 14-bit register. This register is entered with a function 4 or 5 request.

The I/O Memory address in the channel register is forced to a value that is a multiple of four. This is done by forcing the 2 low-order bits of the register content to zero values. This provision allows the maximum data rate to the I/O Processor I/O Memory.

All three register values are cleared to 0 at the end of a block copy. A zero value will be used if a register entry is omitted in the next sequence.

At the end of a read transfer (done flag set), the busy flag is left set if a multiple bit error occurred in the transfer.

MOS : 0§ Clear the channel busy and channel done flags.

MOS : 1 Enter the accumulator content in the channel interface register for the I/O Memory address. The 2 low-order bits are forced to 0.

MOS : 2 Enter the 15 low-order bits of the accumulator content as the 15 high-order bits of the Buffer Memory address. See figure 5-1.

MOS : 3 Enter the 9 low-order bits of the accumulator content as the 9 low-order bits of the Buffer Memory address. See figure 5-1.

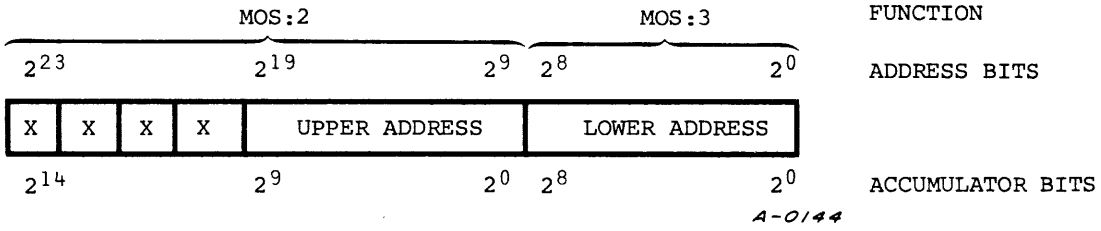


Figure 5-1. Buffer Memory address formation

§ Allow 1 CP before checking busy or done.

MOS : 4[§] Initiate the transfer of the block of data from the Buffer Memory to the I/O Processor I/O Memory. The channel busy flag is set and the channel done flag is cleared by the function request. The 14 low-order bits of the accumulator content at the time of the function request is entered in the channel interface block length register. The channel done flag is set and the channel busy flag is cleared when the block transfer has been completed. If the block length register contains a zero value, the block length is set to 65,536 parcels (full I/O Memory). The busy flag is left set if a multiple-bit error occurred in the transfer. Single-bit errors are automatically corrected.

MOS : 5[§] Initiate the transfer of a block of data from the I/O Processor I/O Memory to the Buffer Memory. The channel busy flag is set and channel done flag is cleared by the function request. The 14 low-order bits of the accumulator content at the time of the function request is entered in the channel interface block length register. The channel done flag is set and the channel busy flag is cleared when the block transfer has been completed.

MOS : 6[§§] Clear the channel interrupt enable flag.

MOS : 7[§§] Set the channel interrupt enable flag.

MOS : 14 Enter the 3 low-order bits of the accumulator content in the control register. This is used for diagnostic purposes only.

2^0 = Disable error correction
2^1 = Disable write check bits
2^2 = Disable refresh

Setting a control bit to 0 enables error correction, write check bits, or refresh.

---

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

Error Handling

When an error occurs, the busy and done flags are both set. A MOS : 0 command must be issued to the channel to clear the busy and done flags before another read or write is initiated.

Buffer Memory Interface Deadstart

The Buffer Memory interface has provisions for deadstarting the I/O Processor. If the I/O processor is master cleared with the deadstart signal set, the I/O Memory address register, the Buffer Memory address register, and the block length register are cleared. This sets up a 65K parcel transfer. A MOS : 4 read Buffer Memory function is initiated when the I/O Processor master clear signal goes to 0. The interrupt enable flag is set (MOS : 7), and when the transfer is complete, the I/O Processor interrupts.

Buffer Memory Interface Dead Dump

The Buffer Memory interface also provides for a dead dump of the I/O Processor. If the I/O Processor is master cleared with the dead dump signal set, the I/O Memory address, Buffer Memory address and the block length registers are cleared. A MOS : 5 Buffer Memory function write is initiated when master clear goes to 0. The interrupt enable flag is cleared (MOS : 6) and no interrupt occurs when the transfer completes.

CHANNEL FOR I/O PROCESSOR INPUT (CH. 6, 10, 12)

The I/O Processor has three input channels for connection to other I/O Processors. These provide a communication link between I/O Processors. These channels have no busy flags. Each channel uses a single 16-bit register to hold the data parcel being transferred. The register is cleared as soon as the data enters the receiving accumulator.

AIA : 0[§] Clear the channel done flag. There is no busy flag for this channel.

AIA : 6[§§] Clear the channel interrupt enable flag.

AIA : 7[§§] Set the channel interrupt enable flag.

AIA : 10 Read the accumulator data of the other I/O Processor into the accumulator of this I/O Processor, and resumes the channel. The data does <u>not</u> remain in the interface register after it has been read.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

CHANNEL FOR I/O PROCESSOR OUTPUT (CH. 7, 11, 13)

The I/O Processor has three output channels that connect to other I/O Processors. These provide a communications link between I/O Processors, and provide the ability to master clear, deadstart, or dead dump another I/O Processor.

A 3-bit control register receives the 3 low-order bits of the sending I/O Processor accumulator and causes the required action. A 16-bit register holds transmitted data until it is loaded into the receiving accumulator, at which time the data register is cleared.

A0A : 0§§ Clear the channel busy and done flags.

A0A : 1 Enter the 3 low-order accumulator bits into the control register. A set bit in the register positions causes the following actions:

Bit	Meaning
2^0	Master clear
2^1	Deadstart
2^2	Dead dump

To perform a deadstart from Buffer Memory, set the master clear and deadstart control bits simultaneously and keep them set for at least 200 nanoseconds. Then clear both bits. The deadstart control bit has no effect without the master clear control bit. The deadstart transfer is initiated with a block length of 65K parcels starting at I/O Memory address 0. In approximately 2 milliseconds the transfer completes and then interrupts the I/O Processor.

To perform a dead dump from I/O Memory to Buffer Memory, set the master clear and dead dump control bits simultaneously and keep them set for at least 200 nanoseconds. Then clear both bits. The dead dump control bit has no effect without the master clear control bit. The dead dump transfer initiates with a block length of 65K parcels starting with I/O Memory address 0 contents going into Buffer Memory address 0. The dead dump completes in approximately 2 milliseconds, but does not interrupt the I/O Processor at completion.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

A0A : 6[§§]	Clear channel interrupt enable flag.
A0A : 7[§§]	Set channel interrupt enable flag.
A0A : 14[§]	Set the channel busy flag and output accumulator data. When data is accepted by receiving I/O Processor the busy flag clears and the done flag sets.

INTERRUPT SEQUENCE

The I/O Processor program is interrupted for service by a monitor program when an interrupt request is present and the system interrupt enable flag is set. The system interrupt enable flag applies to the entire I/O Processor, as contrasted to the channel interrupt enable flags that only apply to the particular channel. The 003 I = 1 instruction sets the system interrupt enable flag. The interruption occurs upon completion of an instruction in the currently executing program.

A jump or exit instruction must be completed before interrupts are actually set. Therefore, interrupts are not actually enabled until the first non-branching instruction is executed.

The interrupt sequence begins by the hardware clearing of the system interrupt enable flag to prevent further interrupts. The address for the next instruction in the interrupted program is stored in the exit stack. The entry in the exit stack is made by first advancing the E register by one count, and then entering the exit stack at the newly pointed position. This is the same sequence which occurs on a return jump execution. The execution of the interrupted program is then suspended and a new program sequence initiated. The address for beginning the monitor program sequence is obtained from the 0 position in the exit stack, without using the E register.

The monitor program, when finished, restores the system interrupt enable flag and exits to the interrupted program as if it were a subprogram call. The last two statements in the monitor program should be the equivalent of the following:

I = 1		I = 1 (Set system interrupt enable flag)
EXIT	or	P = dd. (Exit or jump to interrupted program)

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

When issuing an I=1, the system interrupt enable is delayed until the next non-branch or non-I/O instruction is issued. The instructions that do not enable interrupts are the 40-43 and 70-137. This allows the executive/monitor to get back to the interruptible activity before an interrupt is accepted.

An I=0 instruction should be used at the interrupt handler entrance. If a redundant I=1 is executed and an interrupt occurs before a non-branch or non-I/O instruction is encountered, the interrupt handler will be entered (with interrupts disabled). But interrupts will be re-enabled when the first non-branch or non-I/O instruction is issued within the interrupt handler.

After issuing a command 6 or 7 to any I/O channel, allow 3 clock periods before seeing its effect on system interrupt. (Assuming system interrupts are, or will be, enabled.)

IOP INSTRUCTIONS 6

INSTRUCTION FORMAT

Each I/O Processor instruction occupies one or two 16-bit parcels in the I/O Memory. The instruction consists of two designators and a constant field as illustrated in figure 6-1.

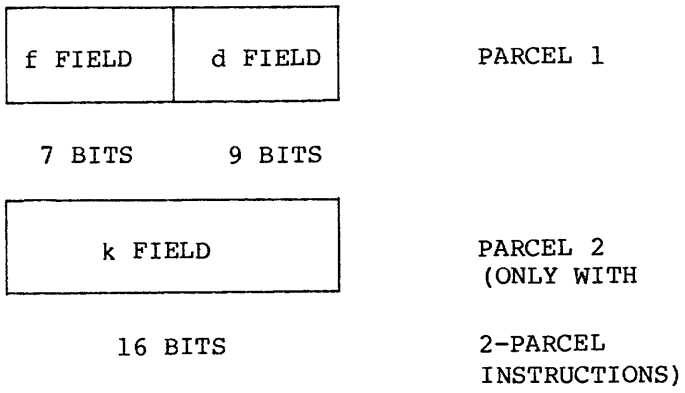


Figure 6-1. Instruction format

The f designator is the instruction function code and specifies which of the instructions in the machine repertoire is intended for execution. The d designator has several uses, depending on the instruction function, but in general specifies where in the machine resources the function of the f designator is to be performed. The d designator may be thought of as a displacement specification and is used in several ways:

- To point to a specific operand register where one is required,
- To specify the amount of the displacement of data in a shift instruction,
- To specify the amount of displacement forward or backward in program code for a branch instruction,
- As an operand value.

The d designator is always treated as a 9-bit positive integer. Small integers may be entered directly into the computation from instructions using the d designator as a constant. Separate instructions are provided to add or subtract this 9-bit constant rather than to consider it as a sign extended quantity. A branch instruction may designate a forward or a backward displacement from the current program location. Separate instructions are provided for the forward and backward jumps using the d designator as a 9-bit displacement magnitude.

Certain instructions use the program parcel immediately following the instruction as a constant field, designated k. These instructions may be considered to be 2-parcel instructions.

INSTRUCTION DESCRIPTIONS

The I/O Processor instruction repertoire is described on the following pages. The mnemonics for each instruction are pseudo machine language statements representing the individual operations performed. Special symbols used in the descriptions are:

>	Shift right
<	Shift left
>>	Shift right circular
<<	Shift left circular
&	Logical product
#	Not equal to
dd	Contents of operand register specified by d field
(dd)	Contents of memory location specified by dd
iod	3-character channel mnemonic, i.e., IOR, PXS,...
B	Contents of register B
(B)	Contents of operand register specified by B register

The following descriptions explain the results of each instruction and show the steps and times involved in executing the instruction.

000 Instruction

PASS

This instruction performs no operation. It is used to fill program fields with null operations where desired.

CP 0 Issue.

001 Instruction

EXIT

The EXIT instruction terminates execution of the current program sequence and returns to the sequence that was suspended in calling this subprogram. The current P register value is discarded. The beginning address for the reinitiated sequence is obtained from the program exit stack at the location currently pointed by E. The value of E is then decremented by 1. The decrementing is blocked and the exit stack boundary flag is set if the value of E was previously 0. The exit stack boundary flag will cause an interrupt of the program sequence for restructuring the content of the program exit stack.

If the EXIT instruction follows a modification of the program exit stack or of the E pointer, at least 4 CPs must elapse between the last modification and the EXIT instruction. Reading the modified value back to the accumulator may be used as the necessary delay.

CP 0 Issue. Transmit exit stack data to P.
Decrement E.

002 Instruction

I = 0

This instruction clears the system interrupt enable flag.

CP 0 Issue. Clear system interrupt enable flag.

003 Instruction

I = 1

This instruction sets the system interrupt enable flag.

When issuing an I = 1, the system interrupts enable is delayed until the next non-branch or non-I/O instruction is issued. Instructions that do not enable interrupts after I = 1 issue are the 040-043 and 070-137 instructions. The delay in setting flag for this instruction allows the interrupt program to re-enable the interrupt mode and then exit to the interrupted program.

If the instruction following the I = 1 is an I = 0, the I = 0 takes precedence and system interrupts are disabled.

CP 0 Issue. Set delay interrupt enable flag.

004 Instruction

A = A > d

This instruction shifts the content of the accumulator and the associated carry flag to the right by d bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. Zero values are entered in the carry flag and propagated to the right as the shift progresses. No shift is performed if the shift count is 0. The accumulator and carry flag are cleared if the shift count is greater than 16 decimal.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit accumulator data and inverted d to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

005 Instruction

A = A < d

This instruction shifts the content of the accumulator and the associated carry flag to the left by d bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. Zero values are entered in the low-order bit positions of the accumulator and are propagated to the left as the shift progresses. Bits shifted from the carry flag are discarded. No shift is performed if the shift count is 0. The accumulator and carry flag are cleared if the shift count is greater than 17_{10}.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit accumulator data and d to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

006 Instruction

A = A >> d

This instruction shifts the content of the accumulator and the associated carry flag to the right in a circular mode by d bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. No bits are discarded in this shift mode. Bits shifted from the right end of the accumulator are returned to the carry flag. No shift is performed if the shift count is 0.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit accumulator data and inverted d to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

007 Instruction

A = A << d

This instruction shifts the content of the accumulator and the associated carry flag to the left in a circular mode by d bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. No bits are discarded in this shift mode. Bits shifted from the carry flag are returned to the low-order bit position in the accumulator. No shift is performed if the shift count is 0.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit accumulator data and d to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

010 Instruction

A = d

This instruction enters the d designator in the accumulator as a 9-bit positive integer and clears the carry flag. The high-order bits are 0.

CP 0 Issue. Transmit d to accumulator.

011 Instruction

A = A & d

This instruction forms the bit-by-bit logical product of the previous accumulator content and the d designator and places the result in the accumulator. The d designator is treated as a 9-bit positive integer. It then clears the carry flag.

The logical product of the previous accumulator content and the d operand from the instruction is formed at the input to the accumulator.

CP 0 Issue. Transmit d and accumulator to accumulator.

012 Instruction

A = A + d

This instruction adds the d designator to the previous accumulator content in the 16-bit twos complement mode. The d designator is treated as a 9-bit positive integer in this addition. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit data to adder. Add mode.

CP 2 Transmit adder result to accumulator.

013 Instruction

A = A - d

This instruction subtracts the d designator from the previous accumulator content in a 16-bit twos complement mode. The d designator is treated as a 9-bit positive integer in this operation. The subtraction is performed by complementing the content of the addend register and adding the result to the previous accumulator content. One is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator during either addition process.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit data to adder. Subtract mode.

CP 2 Transmit adder result to accumulator.

014 Instruction

A = k

This instruction enters a 16-bit constant in the accumulator and clears the carry flag. The constant is obtained from the next sequential parcel in the program field. The next instruction is obtained from the following parcel.

CP 0 Issue. Read next parcel out of instruction stack.

CP 1 Transmit k data to accumulator. The function of CP 1 is delayed if the next parcel of instruction buffer data is not available in the instruction stack.

015 Instruction

A = A & k

This instruction forms the bit-by-bit logical product of the previous accumulator content and a 16-bit constant and places the result in the accumulator. It then clears the carry flag. The constant is obtained from the next sequential parcel in the program field. The next instruction is obtained from the following parcel.

The logical product of the previous accumulator content and the operand constant is formed at the input to the accumulator.

CP 0 Issue. Read the next parcel out of the instruction stack.

CP 1 Transmit k data to accumulator. The function of CP 1 is delayed if the next parcel of instruction buffer data is not available in the instruction stack.

016 Instruction

A = A + k

This instruction adds a 16-bit constant to the previous accumulator content in a twos complement mode. The constant is obtained from the next sequential parcel in the program field. The next instruction is obtained from the following parcel. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Read next parcel out of the instruction stack.

CP 1 Transmit k data to addend register. The function of CP 1 is delayed if the next parcel of instruction buffer data is not available in the instruction stack.

CP 2 Transmit data to adder. Add mode.

CP 3 Transmit adder result to accumulator.

017 Instruction

A = A - k

This instruction subtracts a 16-bit constant from the previous accumulator content in a twos complement mode. The constant is obtained from the next sequential parcel in the program field. The next instruction is obtained from the following parcel. The subtraction is performed by complementing the constant and adding the result to the accumulator content. One is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator in either addition process.

This instruction is redundant in the sense that an equivalent function can be performed with the 016 instruction using a different constant. This instruction is included in the list for completeness in the translational pattern.

CP 0 Issue. Read next parcel out of the instruction stack.

CP 1 Transmit k data to addend register. The function of CP 1 is delayed if the next parcel of instruction buffer data is not available in the instruction stack.

CP 2 Transmit data to adder. Subtract mode.

CP 3 Transmit adder result to accumulator.

020 Instruction

A = dd

This instruction enters the content of operand register d in the accumulator and clears the carry flag.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator.

021 Instruction

A = A & dd

This instruction forms the bit-by-bit logical product of the previous accumulator content and the content of operand register d and places the result in the accumulator. It then clears the carry flag.

The logical product of the previous accumulator content and the operand register content is formed at the input to the accumulator. No additional time is required for this function.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator.

022 Instruction

A = A + dd

This instruction adds the content of operand register d to the previous accumulator content. The addition is performed in the twos complement mode. It complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to addend register.

CP 2 Transmit data to adder. Add mode. The data in the accumulator and addend register is transmitted to the adder in this clock period. The addend register is free in this clock period to accept another operand.

CP 3 Transmit adder result to accumulator.

023 Instruction

A = A - dd

This instruction subtracts the content of operand register d from the previous accumulator content. The subtraction is performed by complementing the subtrahend and adding the result to the accumulator content in ones complement mode. One is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to addend register.

CP 2 Transmit data to adder; subtract mode. The data in the accumulator and addend register is transmitted to the adder in this clock period. The addend register is free in this clock period to accept another operand.

CP 3 Transmit adder result to accumulator.

024 Instruction

dd = A

This instruction stores the accumulator content in operand register d.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit d data to RP and DP.

CP 1 Transmit accumulator data to operand register. The function of CP 1 is delayed if the data is not available in the accumulator during the indicated clock period. In this case, the pointer in RP and DP is held until the accumulator data is available.

025 Instruction

dd = A + dd

This instruction adds the content of operand register d to the previous accumulator content. The addition is performed in a twos complement mode. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process. It then replaces the content of operand register d with the new accumulator content.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit d data to RP and DP.

CP 1 Transmit operand register data to addend register. The RP pointer is discarded in CP 1 and reentered with the same pointer from DP in CP 3. The function at CP 1 is delayed if the data is not available in the accumulator.

CP 2 Transmit data to adder. Add mode.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit accumulator data to operand register.

026 Instruction

dd = dd + 1

This instruction replaces the content of operand register d with the previous content increased by 1. The result is left in the accumulator as well as in the operand register. The carry flag is cleared at the beginning of this operation. One is entered in the accumulator. The content of the operand register enters the addend register and is then added to the accumulator content in a twos complement mode. The carry flag is set if a carry is propagated from the accumulator in the addition process. The result is then returned to the operand register.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit d data to RP and DP.

CP 1 Transmit operand register data to addend register. Enter a +1 in the accumulator.

The RP pointer is discarded in CP 1 and reentered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit copy of accumulator data to operand register.

027 Instruction

dd = dd - 1

This instruction replaces the content of operand register d with the previous content decreased by 1. The result is left in the accumulator as well as in the operand register. The carry flag is cleared at the beginning of this operation. The accumulator bits are forced set. The content of the operand register is entered in the addend register and then added to the accumulator content. The carry flag is set if a carry is propagated from the accumulator in the addition process. The result is then returned to the operand register.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit d data to RP and DP.

CP 1 Transmit operand register data to addend register.
Enter a -1 in the accumulator.

The RP pointer is discarded in CP 1 and reentered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit copy of accumulator data to operand register.

030 Instruction

A = (dd)

This instruction enters the content of an I/O Memory location in the accumulator. The I/O Memory address is obtained from operand register d. It then clears the carry flag.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send Memory Read Reference. The function of CP 2 is repeated until the acceptance signal is received.

CP 3 Acceptance signal from I/O Memory. The function of CP 2 is repeated until the acceptance signal is received.

CP 4 -

CP 5 -

CP 6 Transmit memory data to accumulator.

031 Instruction

A = A & (dd)

This instruction forms the bit-by-bit logical product of the previous accumulator content and the content of an I/O Memory location and places the result if the accumulator. The I/O Memory address is obtained from operand register d. It then clears the carry flag.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send memory read reference. The function of CP 2 is repeated until the acceptance signal is received.

CP 3 Acceptance signal from I/O Memory.

CP 4 -

CP 5 -

CP 6 Transmit memory data to accumulator. The logical product of memory data and the accumulator content is done at the input to the accumulator. No additional time is required for this function.

032 Instruction

A = A + (dd)

This instruction adds the content of an I/O Memory location to the content of the accumulator. The I/O Memory address is obtained from operand register d. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read request to memory.

CP 3 Acceptance signal from I/O Memory.

CP 4 The function of CP 2 is repeated until the acceptance signal is received.

CP 5 -

CP 6 Transmit memory data to addend register.

CP 7 Transmit data to adder. Add mode.

The data from the accumulator and addend register is transmitted to the adder in CP 7. The addend register is free in this clock period to accept another operand.

CP 8 Transmit adder result to accumulator.

033 Instruction

A = A - (dd)

This instruction subtracts the content of an I/O Memory location from the content of the accumulator. The I/O Memory address is obtained from operand register d. The subtraction is performed by complementing the subtrahend and adding the result to the accumulator content in a ones complement mode. A 1 is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator during either addition process.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read request to memory.

CP 3 Acceptance signal from I/O Memory.

CP 4 The function of CP 2 will be repeated until the acceptance signal is received.

CP 5 -

CP 6 Transmit memory data to addend register.

CP 7 Transmit data to adder. Subtract mode.

The data from the accumulator and addend register is transmitted to the adder in CP 7. The addend register is free in this clock period to accept another operand.

The complementing of the addend data is accomplished in the transmission of the data from the addend register to the adder. The addition of +1 to the result is accomplished in this same clock period in the adder.

CP 8 Transmit adder result to accumulator.

034 Instruction

(dd) = A

This instruction replaces the content of an I/O Memory location with the current content of the accumulator. The I/O Memory address is obtained from operand register d.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted or the accumulator data is not available.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Transmit a copy of the accumulator data to bank operand registers. Send write request to memory.

The function of CP 2 is repeated until an acceptance signal is received.

CP 3 Acceptance signal from I/O Memory.

035 Instruction

(dd) = A + (dd)

This instruction replaces the content of an I/O Memory location with its previous content plus the current accumulator content. The I/O Memory address is obtained from operand register d. The addition is performed using the accumulator in a 16-bit twos complement mode. The carry flag is complemented if a carry is propagated from the accumulator in the addition process. The result is left in the accumulator as well as transmitted to the I/O Memory location.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted or the accumulator data is not available.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read requet to memory. The functions of CP 2 is repeated until an acceptance signal is received.

CP 3 Acceptance signal from I/O Memory. The MA data is held from CP 3.

CP 4 -

CP 5 -

CP 6 Transmit memory data to addend register.

CP 7 Transmit data to adder. Add mode.

CP 8 Transmit adder result to accumulator.

CP 9 Transmit MA data to bank address registers. Transmit a copy of the accumulator data to bank operand registers. Send write request to memory.

The function of CP 9 is repeated until an acceptance signal is received.

CP 10 Acceptance signal from I/O Memory.

036 Instruction

(dd) = (dd) + 1

This instruction increments the content of an I/O Memory location by 1. The I/O Memory address is obtained from operand register d. This operation is performed using the accumulator. The accumulator and carry flag are cleared. A +1 is entered in the accumulator. The content of the memory location is entered in the addend register and then added to the accumulator contents. The result is returned to I/O Memory. The result remains in the accumulator. The carry flag is complemented if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read request to memory. The function of CP 2 is repeated until an acceptance signal is received.

CP 3 Acceptance signal from I/O Memory. The MA data is held from CP 3.

CP 4 -

CP 5 -

CP 6 Transmit memory data to addend register. Clear carry flag and enter plus one in accumulator.

CP 7 Transmit data to adder. Add mode.

CP 8 Transmit adder result to accumulator.

CP 9 Transmit MA data to bank address registers. Transmit copy of accumulator data to bank operand registers. Send write request to memory. The function of CP 9 is repeated until the acceptance signal is received.

CP 10 Acceptance signal from I/O Memory.

037 Instruction

(dd) = (dd) - 1

This instruction decrements the content of an I/O Memory location by 1. The I/O Memory address is obtained from operand register d. This operation is performed using the accumulator. The carry flag is cleared. The accumulator bits are forced set. The content of the I/O memory location is entered in the addend register and then added to the accumulator content. The result is returned to the I/O memory location. The result also remains in the accumulator. The carry flag is complemented if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read request to memory. The functions of CP 2 is repeated until an acceptance signal is received.

CP 3 Acceptance signal from I/O Memory. The MA data is held from CP 3.

CP 4 -

CP 5 -

CP 6 Transmit memory data to addend register. Clear carry flag and enter all ones in accumulator.

CP 7 Transmit data to adder. Add mode.

CP 8 Transmit adder result to accumulator.

CP 9 Transmit MA data to bank address registers. Send write request to memory. Transmit copy of accumulator data to bank operand registers.

The function of CP 9 is repeated until the acceptance signal is received.

CP 10 Acceptance signal from I/O Memory.

040 Instruction

C = 1, iod = DN

This instruction forces the carry flag to the same state as the channel d done flag.

Another instruction that would alter the state of the carry flag before CP 4 cannot issue in CP 1 or CP 2. A delay of 1 CP must be inserted if an I/O instruction is issued that alters the Busy or Done flags before the 40 instruction.

Channel 000 is always done. The carry flag can be set by setting d = 000 in this instruction.

CP 0 Issue. Transmit d designator to I/O channels.

CP 1 -

CP 2 -

CP 3 Force state of carry flag.

041 Instruction

C = 1, iod = BZ

This instruction forces the carry flag to the same state as the channel d busy flag.

Another instruction that would alter the state of the carry flag before CP 4 cannot issue in CP 1 or CP 2. A delay of 1 CP must be inserted if an I/O instruction is issued that alters the Busy or Done flags before the 41 instruction.

Channel 000 is never busy. The carry flag can be forced clear by setting d to 000 in this instruction.

CP 0 Issue. Transmit d designator to I/O channels.

CP 1 -

CP 2 -

CP 3 Force state of carry flag.

042 Instruction

C = 1, IOB = DN

This instruction forces the carry flag to the same state as the done flag of the channel specified by the contents of the B register.

Another instruction that would alter the state of the carry flag before CP 4 cannot issue in CP 1 or CP 2. A delay of 1 CP must be inserted if an I/O instruction is issued that alters the Busy or Done flags before the 42 instruction.

This instruction cannot issue if the B register is reserved.

Channel 000 is always done. The carry flag can be forced clear by setting B to 000 in this instruction.

CP 0 Issue. Transmit B designator to I/O channels.

CP 1 -

CP 2 -

CP 3 Force state of carry flag.

043 Instruction

C = 1, IOB = BZ

This instruction forces the carry flag to the same state as the busy flag of the channel specified by the contents of the B register.

Another instruction that would alter the state of the carry flag before CP 4 cannot issue in CP 1 or CP 2. A delay of 1 CP must be inserted if an I/O instruction is issued that alters the Busy or Done flags before the 43 instruction.

This instruction cannot issue if the B register is reserved.

Channel 000 is never busy. The carry flag can be forced clear by setting B to 000 for use by this instruction.

CP 0 Issue. Transmit B designator to I/O channels.

CP 1 -

CP 2 -

CP 3 Force state of carry flag.

044 Instruction

A = A > B

This instruction shifts the content of the accumulator and the associated carry flag to the right by B bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. Zero values are entered in the carry flag and propagated to the right as the shift progresses. No shift is performed if the shift count is 0. The accumulator and carry flag are cleared if the shift count is greater than 17_{10}.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data and inverted B to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

045 Instruction

A = A < B

This instruction shifts the content of the accumulator and the associated carry flag to the left by B bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. Zero values are entered in the low-order bit positions of the accumulator and are propagated to the left as the shift progresses. Bits shifted from the carry flag are discarded. No shift is performed if the shift count is 0. The accumulator and carry flag are cleared if the shift count is greater than 17_{10}.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data and B to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

046 Instruction

A = A >> B

This instruction shifts the content of the accumulator and the associated carry flag to the right in a circular mode by B bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. No bits are discarded in this shift mode. Bits shifted from the right end of the accumulator are returned to the carry flag. No shift is performed if the shift count is 0.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data and inverted B to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

047 Instruction

A = A << B

This instruction shifts the content of the accumulator and the associated carry flag to the right in a circular mode by B bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. No bits are discarded in this shift mode. Bits shifted from the right end of the accumulator are returned to the carry flag. No shift is performed if the shift count is 0.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data and B to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

050 Instruction

A = B

This instruction enters the B register content in the accumulator as a 9-bit positive integer and clears the carry flag. The high-order bits are 0.

CP 0 Issue. Transmit B to accumulator.

051 Instrucion

A = A & B

This instruction forms the bit-by-bit logical product of the previous accumulator content and the B register content, and places it in the accumulator. The B register content is treated as a 9-bit positive integer. The instruction clears the carry flag.

The logical product of the previous accumulator content and the operand from the instruction is formed at the input to the accumulator.

CP 0 Issue. Transmit B and accumulator to accumulator.

052 Instruction

A = A + B

This instruction adds the B register content to the previous accumulator content in the 16-bit twos complement mode. The B register content is treated as a 9-bit positive integer in this addition. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data to adder. Add mode.

CP 2 Transmit adder result to accumulator.

053 Instruction

A = A - B

This instruction subtracts the B register content from the previous accumulator content in a 16-bit twos complement mode. The B register content is treated as a 9-bit positive integer in this operation. The subtraction is performed by complementing the content of the addend register and adding the result to the previous accumulator content. One is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator during either addition.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data to adder. Subtract mode.

CP 2 Transmit adder result to accumulator.

054 Instruction

B = A

This instruction replaces the B register content with the low-order 9 bits of the accumulator content.

This instruction cannot issue until the accumulator sequence has been completed for any previous instructions.

CP 0 Issue. B register is free.

CP 1 Issue + 1 (is accumulator ready).

CP 2 Issue + 2 (transmit accumulator data to B register).

055 Instruction

B = A + B

This instruction adds the content of the B register to the previous accumulator content. The B register content is treated as a 9-bit positive integer in this operation. The addition is performed in a 16-bit twos complement mode. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process and then replaces the B register content with low-order 9 bits of the accumulator content.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data to adder. Add mode.

CP 2 Transmit adder result to accumulator. The addend register is free in this CP.

CP 3 Transmit a copy of the accumulator data to B register.

056 Instrucion

B = B + 1

This instruction replaces the content of the B register with its previous content increased by 1. The result is left in the accumulator as well as in the B register. The carry flag is cleared at the beginning of this operation. One is entered in the accumulator. The content of the B register is then added to the accumulator content in a 16-bit twos complement mode. The B register content is treated as a 9-bit positive integer in this process. The low-order 9 bits of the accumulator content are then returned to the B register.

CP 0 Issue. Transmit B to addend register. Enter a +1 in the accumulator.

CP 1 Transmit data to adder. Add mode. The addend register is free in this CP.

CP 2 Transmit adder result to accumulator.

CP 3 Transmit a copy of the accumulator data to B register.

057 Instruction

B = B - 1

This instruction replaces the content of the B register with its previous content decreased by 1. The result is left in the accumulator as well as in the B register. The carry flag is cleared at the beginning of this operation. The accumulator bits are forced set. The content of the operand register is entered in the XB register and then added to the accumulator content. The B register content is treated as a 9-bit positive integer in this process. The carry flag is complemented if a carry is propagated from the accumulator in the addition process. The low-order 9 bits of the accumulator content are then returned to the B register.

CP 0 Issue. Transmit B to addend register. Enter a -1 in the accumulator.

CP 1 Transmit accumulator data to adder. Add mode. The addend register is free in this CP.

CP 2 Transmit adder result to accumulator.

CP 3 Transmit a copy of the accumulator data to B register.

060 Instruction

A = (B)

This instruction enters the content of operand register B in the accumulator. It then clears the carry flag.

CP 0 Issue. Transmit B data to RP.

CP 1 Transmit operand register data to accumulator.

061 Instruction

A = A & (B)

This instruction forms the bit-by-bit logical product of the previous accumulator content and the content of operand register B and places the result in the accumulator. It then clears the carry flag.

The logical product of the previous accumulator content and the operand register content is formed at the input to the accumulator. No additional time is required for this function.

CP 0 Issue. Transmit B data to RP.

CP 1 Transmit operand register data to accumulator.

062 Instruction

A = A + (B)

This instruction adds the content of operand register B to the previous accumulator content. The addition is performed in a twos complement mode. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit B data to RP.

CP 1 Transmit operand register data to addend register.

CP 2 Transmit data to adder; add mode. The data in the accumulator and addend register is transmitted to the adder in CP 2. The addend register is free in this clock period to accept another operand.

CP 3 Transmit adder result to accumulator.

063 Instruction

A = A - (B)

This instruction subtracts the content of operand register B from the previous accumulator content. The subtraction is performed by complementing the subtrahend and adding the result to the accumulator content in a twos complement mode. A one is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator during either addition process.

CP 0 Issue. Transmit B data to RP.

CP 1 Transmit operand register data to addend register.

CP 2 Transmit data to adder; subtract mode. The data in the accumulator and addend register is transmitted to the adder in CP 2. The addend register is free in this clock period to accept another operand.

CP 3 Transmit adder result to accumulator.

064 Instruction

(B) = A

This instruction stores the accumulator content in operand register B.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit B data to RP and DP.

CP 1 Transmit accumulator data to operand register. The function of CP 1 is delayed if the data is not available in the accumulator during the indicated clock period. In this case, the pointer in RP and DP is held until the accumulator data is available.

065 Instruction

(B) = A + (B)

This instruction adds the content of operand register B to the previous accumulator content. The addition is performed in a twos complement mode. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process. It then replaces the content of operand register B with the new accumulator content.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit B data to RP and DP.

CP 1 Transmit operand register data to addend register. The RP pointer is discarded in CP 1 and reentered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode the function at CP 2 is delayed if the data is not available in the accumulator. The addend register is free in this CP.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit a copy of the accumulator data to operand register.

066 Instruction

(B) = (B) + 1

This instruction replaces the content of operand register B with its previous content increased by 1. The result is left in the accumulator as well as in the operand register. The carry flag is cleared at the beginning of this operation. One is entered in the accumulator. The content of the operand register entered in the addend register and is then added to the accumulator content in a twos complement mode. The carry flag is complemented if a carry is propagated from the accumulator in the addition process. The result is then returned to the operand register.

This instruction cannot issue if the DP register contains a pointer from a previous intruction.

CP 0 Issue. Transmit B data to RP and DP.

CP 1 Transmit operand register data to addend register. Enter a plus one in the accumulator.

The RP pointer is discarded in CP 1 and re-entered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode. The addend register is free in this CP.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit copy of accumulator data to operand register.

067 Instruction

(B) = (B) - 1

This instruction replaces the content of operand register B with its previous content decreased by 1. The result is left in the accumulator as well as in the operand register. The carry flag is cleared at the beginning of this operation. The accumulator bits are forced set. The content of the operand register is entered in the addend register and then added to the accumulator content. The carry flag is complemented if a carry is propagated from the accumulator in the addition process. The result is then returned to the operand register.

This instruction cannot issue if the DP register contains a pointer from a previous intruction.

CP 0 Issue. Transmit B data to RP and DP.

CP 1 Transmit operand register data to addend register. Enter an all ones value in the accumulator.

The RP pointer is discarded in CP 1 and re-entered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode. The addend register is free in this CP.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit copy of accumulator data to operand register.

070 Instruction

P = P + d

This instruction terminates the current program sequence and begins a new sequence. The initial address for the new sequence is obtained by adding the d designator to the address of the current instruction. The d designator is treated as a 9-bit positive integer and is added in a 16-bit twos complement mode. The accumulator content and carry flag are not altered in this process.

The issue of further instruction is blocked until the branch mode is resolved.

CP 0 Issue. Transmit P data to accumulator. Transmit d data to addend register.

CP 1 Transmit accumulator data to adder. Add mode.

CP 2 Transmit adder result to P register. The first instructon in the new program sequence is transmitted from the instruction stack to the II register in CP 2 if the branch is within the range of the stack. That intruction may issue at CP 4.

CP 3 If the branch is out of stack, a fetch request at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

071 Instruction

P = P - d

This instruction terminates the current program sequence and begins a new sequence. The initial address for the new sequence is obtained by subtracting the d designator to the address of the current instruction. The d designator is treated as a 9-bit positive integer and is subtracted in a 16-bit twos complement mode. The accumulator content and carry flag are not altered in this process.

The issue of further instruction is blocked until the branch mode is resolved.

CP 0 Issue. Transmit P data to accumulator. Transmit d to addend register.

CP 1 Transmit accumulator data to adder. Subtract mode.

CP 2 Transmit adder result to P register. The first instruction in the new program sequence is transmitted from the instruction stack to the II register in CP 2 if the branch is within the range of the stack. This intruction may issue at CP 4.

CP 3 If the branch is out of stack, a fetch request at the new P address will be generated. This action may be delayed m CPs due to previous internal fetch requests.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

072 Instruction

R = P + d

This instruction suspends execution of the current program sequence and calls a subprogram for execution by the following actions. It advances the value of E by 1 and stores the address of the next sequential instruction of this program sequence in the program exit stack. Then, it begins executing a new program sequence. The initial address for the new sequence is obtained by adding the d designator to the address of the current instruction. The d designator is treated as a 9-bit positive integer and is added in a 16-bit twos complement mode. The accumulator content and carry flag are not altered in ths process.

CP 0 Issue. Transmit P data to accumulator. Transmit d to addend register.

CP 1 Transmit accumulator data to adder. Add mode. Advance E by 1.

CP 2 Transmit P+1 data to program exit stack. Transmit adder result to P register.

CP 3 If the branch is out of stack, a fetch reqest at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

073 Instruction

R = P - d

This instruction suspends execution of the current program sequence and calls a subprogram for execution by the following actions. It advances the value of E by 1 and stores the address of the next sequential instruction of this program sequence in the program exit stack. Then, it begins executing a new program sequence. The initial address for the new sequence is obtained by subtracting the d designator from the address of the current instruction. The d designator is treated as a 9-bit positive integer and is subtracted in a 16-bit twos complement mode. The accumulator content and carry flag are not altered in ths process.

CP 0 Issue. Transmit P data to accumulator. Transmit d to addend register.

CP 1 Transmit accumulator data to adder. Subtract mode. Advance E by 1.

CP 2 Transmit P+1 data to program exit stack. Transmit adder result to P register.

CP 3 If the branch is out of stack, a fetch request at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

074 Instruction

P = dd

This instruction terminates the current program sequence and begins a new sequence. The initial address for the new sequence is obtained from operand register d.

The program fetch request flag is set if the content of operand register d is 0. The issue of further instructions is blocked until the first instruction of the new sequence is in the II register.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator. Enter 0 in the addend register.

CP 2 Transmit accumulator data to adder. Add mode.

CP 3 Transmit adder result to P register. A fetch request at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

075 Instruction

P = dd + k

This instruction terminates the current program sequence and begins a new sequence. The initial address for the new sequence is obtained by adding the content of operand register d to the next parcel of the current program sequence. The addition is performed in a 16-bit twos complement mode. The content of the accumulator and carry flag are not altered in this process.

The program fetch request flag is set if the content of operand register d is 0. The issue of further instructions is blocked until the first instruction of the new program sequence is not in the II register in CP 1.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator. Transmit k data to addend register.

CP 2 Transmit accumulator data to adder. Add mode.

CP 3 Transmit adder result to P register. A fetch request at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

076 Instruction

R = dd

This instruction suspends execution of the current program sequence and calls a subprogram for execution by the following actions. It advances the value of E by 1 and stores the address of the next sequential instruction of this sequence in the program exit stack. It then begins executing a new program sequence. The initial address for the new sequence is obtained from operand register d.

The exit stack boundary flag is set if the advanced E value is 14. The program fetch request flag is set if the content of operand register d is 0. The issue of further instructions is blocked until the first instruction of the new program sequence is in the II register.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator. Enter a 0 value in addend register.

CP 2 Transmit accumulator data to adder. Add mode. Advance E by 1.

CP 3 Transmit P+1 data to exit stack. Transmit adder result to P register. A fetch request at the new P address is generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new intruction.

077 Instruction

R = dd + k

This instruction suspends execution of the current program sequence and calls a subprogram for execution by the following actions. It advances the value of E by 1 and stores the address two greater than the address of the current instruction in the program exit stack. Then it begins a new program sequence. The initial address for the new sequence is obtained from operand register d to the next parcel of the current program sequence. The addition is performed in a 16-bit twos complement mode. The content of the accumulator and carry flag are not altered in this process.

The exit stack boundary flag is set if the advanced E value is 14. The program fetch request flag is set if the content of operand register d is 0. The issue of further instructions is blocked until the first instruction of the new program sequence is in the II register.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator. Transmit k data to addend register.

The entry of the k data in the addend register is delayed if the next parcel of the program sequence is not in the II register in CP 1.

CP 2 Transmit accumulator data to adder. Add mode. Advance E by 1.

CP 3 Transmit P+1 data to exit stack. Transmit adder result to P register. A fetch requst at the new P address is generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

100-137 Conditional Branch Instructions

Instructions 100 through 137 are branch instructions that jump to a new program sequence only if a branch condition is met. There are eight branch modes, which are represented in the unconditional form by instructions 070 through 077. All possibilities of these eight modes are combined with four branch criteria to form the set of instructions 100 through 137. The branch criteria are as follows:

------, C = 0
------, C = 1
------, A = 0
------, A # 0

The first of these branch conditions, C = 0, causes the branch to be taken if the carry flag is 0. If the carry flag is set, the current program sequence is continued.

The second branch condition, C = 1, is the complement of the first. The branch is taken if the carry flag is set. The current sequence is continued if the carry flag is 0.

The third branch condition, A = 0, causes the branch to be taken if the accumulator content is 0. If the accumulator content is nonzero, the current sequence is continued.

The final branch condition, A # 0, is the complement of the third. The branch is taken if the accumulator content is nonzero. The current sequence is continued if the accumulator content is 0.

The timing of the above sequences is the same as the timing of the corresponding instruction in the unconditional mode, if the branch is taken. Formation of the branch condition requires 1 CP after the accumulator has received the desired data. The issue of the next instruction is delayed until the branch criterion is available. If the branch criterios is available in CP 0 and the branch is not taken based on that criterion, the next instruction in the current program sequence may issue in the next clock period.

140-177 I/O Channel Instructions

Instructions 140 through 177 allow I/O processor control of the I/O channel activity. The d designator in instructions 140 through 157 specifies which I/O channel is addressed. In the 160 through 177 instructions, the content of the B register specifies the channel. The low-order 4 bits of the function code for the instruction are sent to the channel interface control along with a go function signal. This 4-bit code is then interpreted by the channel interface control circuits in a manner unique to that channel.

This series of instructions may provide the accumulator data to the interface and the data coming back from the interface may replace the present accumulator contents. Instructions 150 through 153 and 170 through 173 read a 16-bit quantity into the accumulator. This quantity is whatever value the channel interface provides as a result of its interpretation of the channel function. This data transfer is defined in the description for each individual channel. Instructions 140 through 177 may transfer data from the accumulator to the channel interface.

None of the I/O channel instructions involves any significant delay in the execution of the program sequence. There is no mechanism for the I/O channel control to delay execution of further instructions as a result of interpreting the 4-bit code. Delays in executing program functions must be programmed through sampling of the channel busy and done flags or through the equivalent use of the interrupt mechanism.

After issuing any comment to the I/O channels, allow 1 CP (by issuing a pass instruction, or other instruction) before checking the channel busy or done flags. One CP should also be allowed after any : 6 or : 7 I/O instruction (modifying channel interrupt flag) before checking for the channel interrupt number (IOR : 10).

INTERFACES 7

INTRODUCTION

Interfaces are required to adapt the I/O Processor to peripheral devices to take advantage of its capabilities. The main purposes of an interface are buffering data, generating control signals for the peripheral device, and possibly multiplexing several devices into the same I/O Processor channel. This section describes the characteristics of the interfaces, gives a table of currently used functions for the interfaces, and describes operational characteristics of the interfaces.

INTERFACE CHARACTERISTICS

The I/O Processor provides for 40 I/O channels. These channels are addressed by the d designator in the program instruction or by the B register contents. Data may be transferred from the I/O Processor accumulator to an interface register or from an interface register to the accumulator. I/O Memory ports may be used for block transfers of data into or out of I/O Memory. Data transfers and channel interface actions are a function of each interface logic control.

Each interface may interpret up to 16 function signals from the I/O Processor program. These functions are generated by instructions 140 through 177. Interpretation of each function is specifically designated by each interface. However, three of the function codes are fairly common among interfaces and are described below.

iod : 0 or IOB : 0.

This function clears the channel busy and done flags and places the channel in an idle status.

iod : 6 or IOB : 6.

This function clears the channel interrupt flag for the associated channel, which blocks any further interrupt requests from that channel.

iod : 7 or IOB : 7.

This function sets the channel interrupt enable flag for the associated channel and enables the interrupt requests from that channel.

Each channel interface provides for a busy flag. This flag is normally set during the active period of the channel and cleared during an idle period. The setting and clearing of this flag depends on the channel interface interpretation of the 16 function codes. The channel busy flag may be sensed by the I/O Processor program through execution of the 041 and 043 instructions.

Each channel interface provides for a done flag. This flag is normally used to signal the I/O Processor program when some step of the channel activity has reached a point where program action is required. Setting and clearing of the flag is normally a function of the interface hardware, but the program may set or clear the flags for special purposes. The program may sense the state of this flag through the 040 and 042 instructions. An interrupt is normally generated by the interface hardware when the channel done flag is set and the channel interrupt enable flag is also set. The system must have interrupts enabled to process the interrupt.

INTERFACE FUNCTION CODES

Table 7-1 lists all the currently supported peripheral devices and briefly explains each function code interpretation that has been implemented. The mnemonic shown is for A Programming Machine Language, APML.

Table 7-1. Interface functions

Device	Mnemonic	Function
DISK STORAGE UNIT (DKA-DKP)	DKA : 0	Clear the channel control
	DKA : 1	Select mode or request status
	DKA : 2	Read data into I/O Memory
	DKA : 3	Write data from I/O Memory
	DKA : 4	Select a new head group
	DKA : 5	Select a new cylinder
	DKA : 6	Clear the channel interrupt enable flag
	DKA : 7	Set the channel interrupt enable flag
	DKA : 10	Read I/O Memory current address
	DKA : 11	Read status response
	DKA : 14	Enter I/O Memory beginning address
	DKA : 15	Status response register diagnostic
CONSOLE KEYBOARD (TIA,TIB,TIC...)	TIA : 0	Clear the channel done flag
	TIA : 6	Clear the channel interrupt enable flag
	TIA : 7	Set the channel interrupt enable flag
	TIA : 10	Read data into accumulator and clear done flag
CONSOLE DISPLAY (TOA,TOB,TOC...)	TOA : 0	Clear the channel busy and done flags
	TOA : 6	Clear the channel interrupt enable flag
	TOA : 7	Set the channel interrupt enable flag
	TOA : 14	Send accumulator data to display
EXPANDER CHASSIS	EXB : 0	Idle the channel
	EXB : 1	Data input from A register (DIA)
	EXB : 2	Data input from B register (DIB)
	EXB : 3	Data input from C register (DIC)
	EXB : 4	Read busy/done flag, interrupt number
	EXB : 5	Load device address
	EXB : 6	Send interface mask (MSKO)
	EXB : 7	Set interrupt mode
	EXB : 10	Read data bus status
	EXB : 11	Read status 1
	EXB : 13	Read status 2
	EXB : 14	Data output to A register (DOA)
	EXB : 15	Data output to B register (DOB)
	EXB : 16	Data output to C register (DOC)
	EXB : 17	Send control

Table 7-1. Interface functions (continued)

Device	Mnemonic	Function
INPUT FROM CPU I/O CHANNEL (CIA,CIB,CIC...)	CIA : 0	Clear channel
	CIA : 1	Enter I/O Memory address, start input
	CIA : 2	Enter parcel count
	CIA : 3	Clear channel parity error flags
	CIA : 4	Clear ready waiting flag
	CIA : 6	Clear interrupt enable flag
	CIA : 7	Set interrupt enable flag
	CIA : 10	Read I/O Memory address
	CIA : 11	Read status (ready waiting, parity errors)
OUTPUT TO CPU I/O CHANNEL (COA,COB,COC...)	COA : 0	Clear channel
	COA : 1	Enter I/O Memory address
	COA : 2	Enter parcel count
	COA : 3	Clear error flag
	COA : 4	Set/clear external control signals
	COA : 6	Clear interrupt enable flag
	COA : 7	Set interrupt enable flag
	COA : 10	Read I/O Memory address
	COA : 11	Read status (4-bit channel data, error)
INPUT FROM CPU MEMORY CHANNEL (HIA,HIB,HIC...)	HIA : 0	Clear channel busy, done flags
	HIA : 1	Enter I/O Memory address
	HIA : 2	Enter upper Central Memory address
	HIA : 3	Enter lower Central Memory address
	HIA : 4	Read Central Memory, enter block length
	HIA : 6	Clear interrupt enable flag
	HIA : 7	Set interrupt enable flag
	HIA : 14	Enter diagnostic mode
OUTPUT TO CPU MEMORY CHANNEL (HOA,HOB,HOC...)	HOA : 0	Clear channel busy, done flags
	HOA : 1	Enter I/O Memory address
	HOA : 2	Enter upper Central Memory address
	HOA : 3	Enter lower Central Memory address
	HOA : 5	Write Central Memory, enter block length
	HOA : 6	Clear interrupt enable flag
	HOA : 7	Set interrupt enable flag
	HOA : 14	Enter diagnostic mode

Table 7-1. Interface functions (continued)

Device	Mnemonic	Function
ERROR LOGGING CHANNEL (ERA,ERB,ERC...)	ERA : 0	Idle channel
	ERA : 6	Clear interrupt enable flag
	ERA : 7	Set interrupt enable flag
	ERA : 10	Read error status
	ERA : 11	Read error information (first parameter)
	ERA : 12	Read error information (second parameter)
	ERA : 13	Read error information (third parameter)
BLOCK MULTIPLEXER CHANNEL (BMA,BMB,BMC...)	BMA : 0	Clear channel control
	BMA : 1	Send reset functions
	BMA : 2	Channel command
	BMA : 3	Read request-in address
	BMA : 4	Asynchronous I/O
	BMA : 5	Delay counter diagnostic
	BMA : 6	Clear channel interrupt enable flag
	BMA : 7	Set channel interrupt enable flag
	BMA : 10	Read I/O Memory address
	BMA : 11	Read byte count
	BMA : 12	Read status
	BMA : 13	Read input tags
	BMA : 14	Enter I/O Memory address
	BMA : 15	Enter byte count
	BMA : 16	Enter device address
	BMA : 17	Enter output tags

DISK STORAGE UNIT CHANNEL

Each Buffer I/O Processor and Disk I/O Processor may have up to 16 channels connected to DD-29 Disk Storage Units. Each unit may operate independently of the other units and all may be transferring data at the same time. APML mnemonics DKA through DKP indicate these channels. The function requests for the first channel are summarized below.

DKA : 0§	Clear the channel control
DKA : 1§	Select mode or request status
DKA : 2§	Read data into I/O Memory
DKA : 3§	Write data from I/O Memory
DKA : 4	Select a new head group
DKA : 5	Select a new cylinder§
DKA : 6§§	Clear the channel interrupt enable flag
DKA : 7§§	Set the channel interrupt enable flag
DKA : 10	Read I/O Memory current address
DKA : 11	Read status response
DKA : 14	Enter I/O Memory beginning address
DKA : 15	Status response register diagnostic

DISK STORAGE UNIT CHARACTERISTICS

The DD-29 Disk Storage Unit consists of 40 rotating disk surfaces with a read/record head on each surface. The period of disk rotation is 16.6 milliseconds. The heads are moved simultaneously to one of 823 disk cylinders by means of a servomechanism. Positioning time from one cylinder to another varies from 15 milliseconds to 80 milliseconds depending on the distance the head assembly must travel.

---

§ Valid channel busy and channel done flags cannot be read until 1 CP after this function is issued.

§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

Within each disk cylinder the 40 read/record heads are divided into ten groups. Each head group then reads or records 4 bits of data in parallel. The selection of a new head group requires 6 microseconds. The recording surface available to each head group is called a disk track. This is the basic storage unit reserved by the operating system. A flaw on the disk surface requires that a track be removed from the available resources in the track reservation table for the system.

Within each disk track are 18 sectors in which data may be recorded and read back. The data in one sector is called a data block and consists of 2048 parcels of I/O Processor data plus verification and error correction data. Data may be transferred between the I/O Memory and the disk surface only in blocks of this fixed size.

DSU DATA SEQUENCE PATTERN

The data recorded in a sector of a disk track consists of a number of parts as shown in figure 7-1. The numbers below each segment in the figure are the total bits of all four heads, for the segment.

	GAP	PREAMBLE	SYNC	ID	DELAY	PREAMBLE	SYNC	512 WORDS	CRC	POSTAMBLE	
INDEX	720	912	8	24	304	912	8	32,768	128	24	BITS

Figure 7-1. DSU data sequence pattern

The total number of bits in the above figure is 35,808. This is the portion of a disk track assigned to a sector. An additional gap after the last sector has 576 bits. The total number of bits in a disk track is 645,120.

The bit positions assigned to the angular locations on the disk surface are determined by an index mark and a servo clock. The index mark is a unique mechanical mark on the rotating mechanism which provides a pulse once per disk revolution. This pulse clears a counter which then counts servo clock pulses to define the remainder of the disk timing. Servo pulses are also derived mechanically from the rotating mechanism. These pulses occur every 12 bit positions. The clock used for recording data on the disk surface is obtained by a frequency multiplier. The index mark begins the data sequence pattern listed above for sector 0. The beginning location for the other sectors is determined by the servo counter. These begin every 746 servo pulses or 8952 bit positions.

The data sequence pattern for a sector, as listed above, is recorded in two separate processes. The sector identification (ID) word, which appears as 6 bits under each recording head, is recorded on each new set of disk surfaces and is not modified in normal use of the disk storage unit. The data block and cyclic checkword are recorded with each disk write function in a normal operation. Associated with each of these recordings is a preamble, sync, and postamble which are a necessary part of the recording and reading process.

The write heads are turned on for a normal disk block write function during the intersector gap. The writing begins with the second group of preamble and sync bits which are sequenced by the disk control circuits. The data from the I/O Memory is then recorded in a block of 32,768 bits. This is followed by a 128-bit cyclic redundancy checkword which was generated from the data by a Fire code generator. The write heads are turned off after the 24-bit postamble.

SECTOR IDENTIFICATION WORD

The sector identification (ID) word for each sector is 24 bits. It is composed of a cylinder number, a head-group number, a sector number and four parity bits. The ID format is shown in figure 7-2.

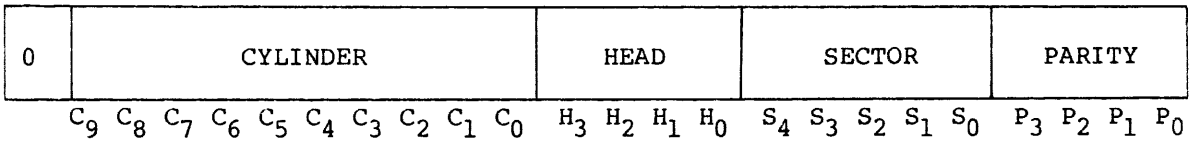


Figure 7-2. Sector ID format

The parity bits in the sector identification word each protect a cross-section of the other 20 bits of the ID. The assignment of parity bits to groups of ID bits is shown in table 7-2.

I/O MEMORY ADDRESS REGISTER

The I/O Memory address register is both an interface input register and an interface output register for the channel. The beginning address for a block of disk data is entered in the register by the processor before issuing a read or write function. This address is restricted to values which are a multiple of four because of the burst mode used in moving data into and out of the I/O Memory. The low-order 2 bits of the address are forced to 0 in the register. If the processor enters bit values other than 0 in these positions, these values are discarded.

Table 7-2. Sector ID parity bit assignments

Parity Bits	$p^0/2^0$	$p^1/2^1$	$p^2/2^2$	$p^3/2^3$
Data Bits	$C^7/2^{20}$ $C^3/2^{16}$ $H^3/2^{12}$ $S^4/2^8$ $S^0/2^4$	$C^8/2^{21}$ $C^4/2^{17}$ $C^0/2^{13}$ $H^0/2^9$ $S^1/2^5$	$C^2/2^{22}$ $C^5/2^{18}$ $C^1/2^{14}$ $H^1/2^{10}$ $S^2/2^{26}$	$0/2^{23}$ $C^6/2^{19}$ $C^2/2^{15}$ $H^2/2^{11}$ $S^3/2^{27}$

C = Cylinder, H = Head, S = Sector, P = Parity

The address in the I/O Memory address register is increased by a count of four as each burst of four words is transferred to or from the I/O Memory. This address may be monitored by the I/O Processor using the DKA : 10 function. The address may be used without a new entry from the processor after completion of a data transfer. In this case the beginning address for the next block of disk data will be the next sequential storage address. For a disk write the I/O Memory address will be left pointing four parcels beyond the last address of the buffer.

STATUS RESPONSE REGISTER

The status response register is used for the specific response requested by a function 1 request and also for the implied response of function 5. Details of these functions are listed under the appropriate headings.

DKA : 0 - CLEAR CHANNEL

Clear the channel busy and channel done flags. No parameters are required for this function. This function is not interlocked with any disk sequence which may be in process.

DKA : 1 - SELECT MODE

This function request allows the processor to select a mode for the disk storage unit or to request status information from the interface. The content of the accumulator at the time of the function request is used as a selecting parameter. The categories of parameter values are summarized in table 7-3.

Table 7-3. DKA : 1 parameters

Value	Meaning
000xxx	Release Unit
001xxx	Reserve Unit
002xxx	Clear fault flags
003xxx	Return to zero cylinder
004xxx	Select margin conditions
005xxx	Read sector number
006xxx	Read error flags
007xxx	Read disk status

Parameter 000xxx - Release Unit

This function request sets the channel busy flag and clears the channel done flag. No other flag request can be made on a particular DSU until a DKA : 1 request has been issued for that unit. After a few microseconds, the interface clears the channel busy flag and sets the channel done flag. The disk storage unit is released from the reservation on that port and is free for reservation on the other port.

Parameter 001xxx - Reserve Unit

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later, the interface clears the channel busy flag and sets the channel done flag. The DSU is reserved for the requesting I/O Processor if it is not currently reserved by another device using the DSU access port. The function does not automatically return a status response. The I/O Processor must issue a separate function 1 status request to determine whether the reservation was accepted. The reserve unit function automatically selects head group 0. The unit must be reserved before it will recognize a read function (DKA : 2), a write function (DKA : 3), or a select cylinder function (DKA : 5). All other functions may be issued to an unreserved unit, unless the DSU is reserved on another access port.

Parameter 002xxx - Clear Fault Flags

This function 1 request with an 002xxx parameter sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. The fault conditions stored in fault registers in the interface and in the DSU are all cleared.

Parameter 003xxx - Return to Zero Cylinder

This function request sets the channel busy flag and clears the channel done flag. The read/write heads are positioned to cylinder 000 as if the DSU were just powered up. The time for positioning the heads depends upon the distance to be traveled, and may extend to 500 milliseconds for an 822-cylinder move. When the positioning is completed, the interface clears the channel busy flag and sets the channel done flag.

Parameter 004xxx - Select Cylinder Margin

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the disk interface clears the channel busy flag and sets the channel done flag. The read/write heads are moved slightly away from the normal cylinder center to attempt reading data which cannot be recovered using normal head positioning. The amount to be offset is determined by the low-order 5 bits in the function parameter. Each unit of the 5-bit value offsets the heads 25 microinches (0.64 micrometers). Bit 2^5 of the parameter defines the direction of the offset: a 1 indicates an offset toward the center of the disk and a 0 specifies a move away from the center of the disk. The nominal cylinder width is 2200 microinches (5600 micrometers), and a nominal center-to-center cylinder spacing spacing is 2600 microinches (6600 micrometers). The offset position is maintained until the next positioning function is received, either another margin select, or a new cylinder select. A following cylinder select automatically cancels the margin select and centers the heads over the new cylinder. The direction and amount of offset active is contained in the DSU offset register, and can be inspected by a DKA : 1, parameter 007002 command.

Parameter 005xxx - Read Sector Number

This function request sets the channel busy flag and clears the channel done flag. About 10 I/O Processor CPs later the interface clears the channel busy flag and sets the channel done flag. At this time the sector number of the sector currently under the read/write heads is loaded into the status response register in the interface. A following function 11 to the interface will read the status response register to the I/O Processor accumulator.

The sector number is not read from the disk. Instead, an interface counter operates from the DSU servo clock, tracking the sector number. The counter is updated at the start of the 180-frame intersector gap, and it is cleared by the index mark.

Parameter 006xxx - Read Error Flags

Issuing this function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the status response register on the interface receives data from the fault registers in the DSU and in the interface. This data remains in the status response register until it is replaced by data from another function request. Figure 7-3 shows the bit assignments for the error flags, and table 7-4 explains each error condition.

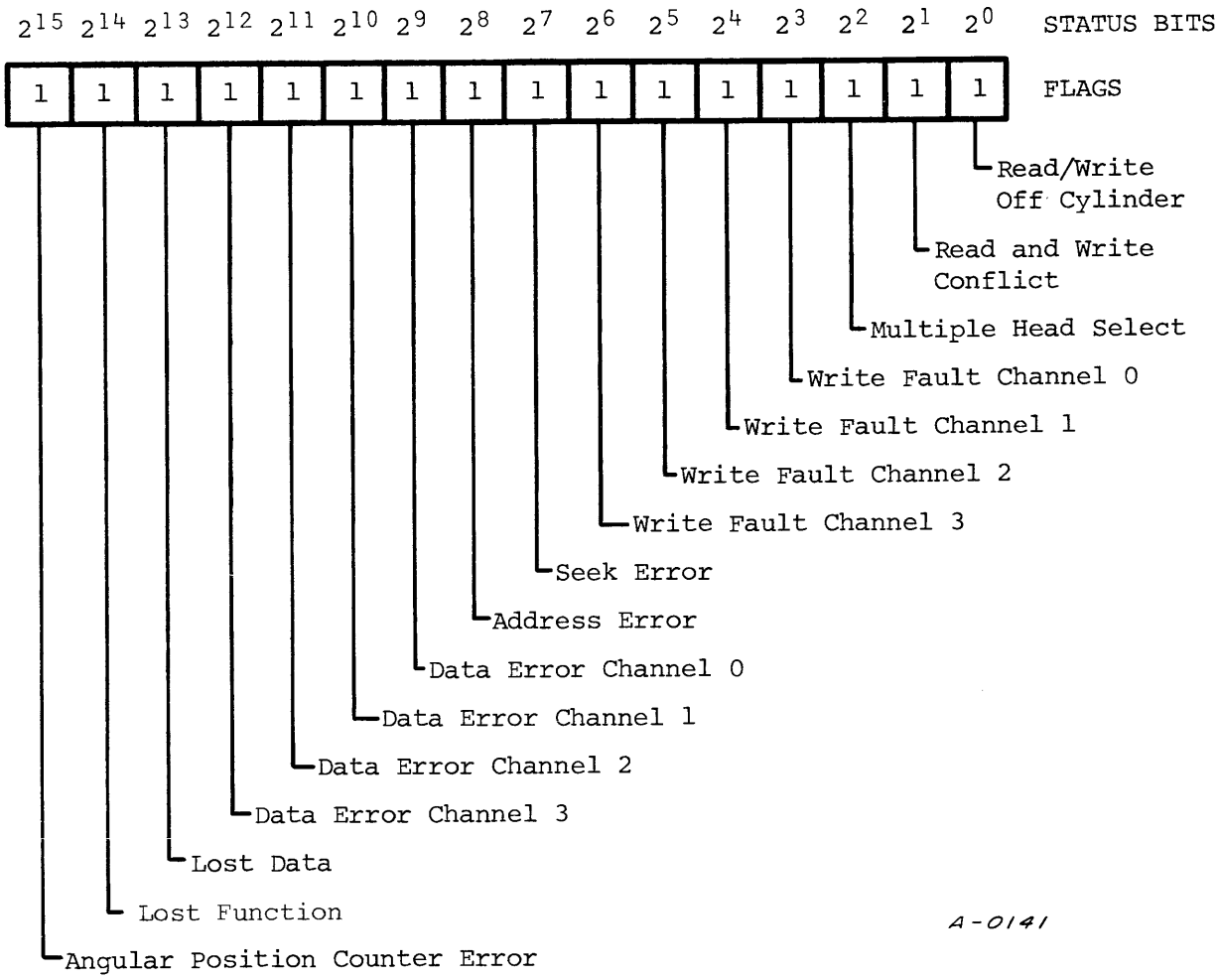


Figure 7-3. Status response error flags

Table 7-4. Parameter 006 error flags

Bit	Name	Meaning
2^0	Read/write off cylinder	An attempt was made to read or write data when the read/write heads were still in motion on a change of cylinder.
2^1	Read and write conflict	An attempt was made to strobe data simultaneously early and late, or an attempt was made to write data with the cylinder margin offset, or the unit attempted to read and write at the same time.
2^2	Multiple head select	More than 4 heads were selected simultaneously.
2^3	Write fault channel 0	A failure occurred associated with the recording head for channel 0.
2^4	Write fault channel 1	A failure occurred associated with the recording head for channel 1.
2^5	Write fault channel 2	A failure occurred associated with the recording head for channel 2.
2^6	Write fault channel 3	A failure occurred associated with the recording head for channel 3.
2^7	Seek error	A failure occurred in moving the read/write heads to a new cylinder number.
2^8	Address error	The disk storage unit received a head group select for a group number greater than 11_8, or a cylinder select for a cylinder number greater than 1466 octal, or a margin selection when the disk was not on cylinder, or a cylinder select when the disk was not on cylinder.
2^9	Data error channel 0	An error occurred in the channel 0 data during the last read operation.
2^{10}	Data error channel 1	An error occurred in the channel 1 data during the last read operation.
2^{11}	Data error channel 2	An error occurred in the channel 2 data during the last read operaton.

Table 7-4. Parameter 006 error flags (continued)

Bit	Name	Meaning
2^{12}	Data error channel 3	An error occurred in the channel 3 data during the last read operation.
2^{13}	Lost data	The data transfer between the I/O Memory and the deskewing buffers did not keep up with the disk read/write transfer in a read or write operation.
2^{14}	Lost function	A function was received before a previous function completed.
2^{15}	Angular position counter error	The index mark from the disk storage unit was received in the middle of a sector.

Parameter 007000 - Read Cylinder Register

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the interface status register receives from the DSU the currently selected cylinder number. The cylinder number occupies the 10 low-order bits of the status register. This data remains in the status register until replaced by another function request.

Parameter 007001 - Read Head Register

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the interface status register receives from the DSU the currently selected head group number, if the DSU is reserved to the processor. The head number goes in the 4 low-order register bits, and bit 2^5 is a 1 to show reservation to the requesting processor. Bit 2^6 indicates the DSU capacity is 600 Mbytes. The register value has a range of 40_8-51_8 for the 10 head groups. A zero word returned indicates the DSU is not reserved to the requesting processor. Status data remains in the register until another function request.

Parameter 007002 - Read Margin/Difference Register

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the interface status response register receives from the DSU either the currently selected offset margin or the difference between the present position of the heads and the final cylinder position during a seek. If the last function request was an offset margin selection, the status is the ones complement of the offset number selected by that function. The 5 low-order bits on the status register hold the offset number, and bit 2^5 shows the offset direction. Bit 2^5 is set to 1 if the offset is toward the center of the disk. See figure 7-4.

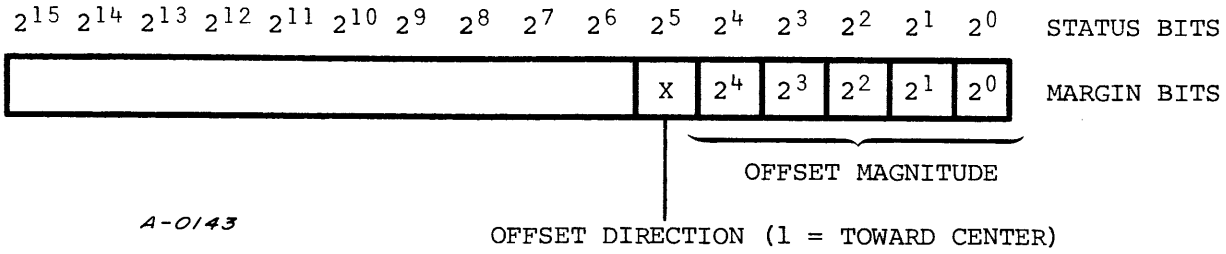
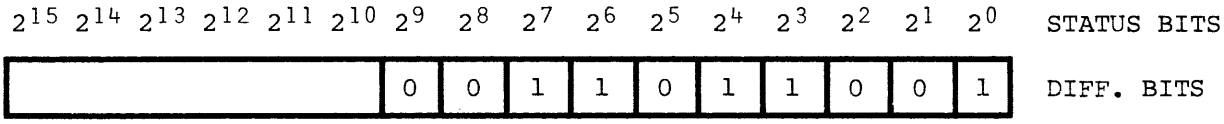


Figure 7-4. Offset margin status word

If the previous function request was a select cylinder, the interface status register receives the ones complement of the number of cylinder positions yet to be crossed before reaching the desired cylinder. The register contains 1777_8 when the heads are positioned at the desired cylinder. The difference number goes into the least significant 10 bits of the status register. See figure 7-5.



Example: 294 = 0100100110 tracks to go
1011011001 one's complement to register

Figure 7-5. Difference register example

Parameter 007003 - Read Interlock Register

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the interface status response register receives from the DSU the contents of the interlock register. Eight interlock flags are placed in the low-order bit positions of the status response register. The interlock flags are shown in figure 7-6. Ones indicate fault conditions. Table 7-5 explains each flag.

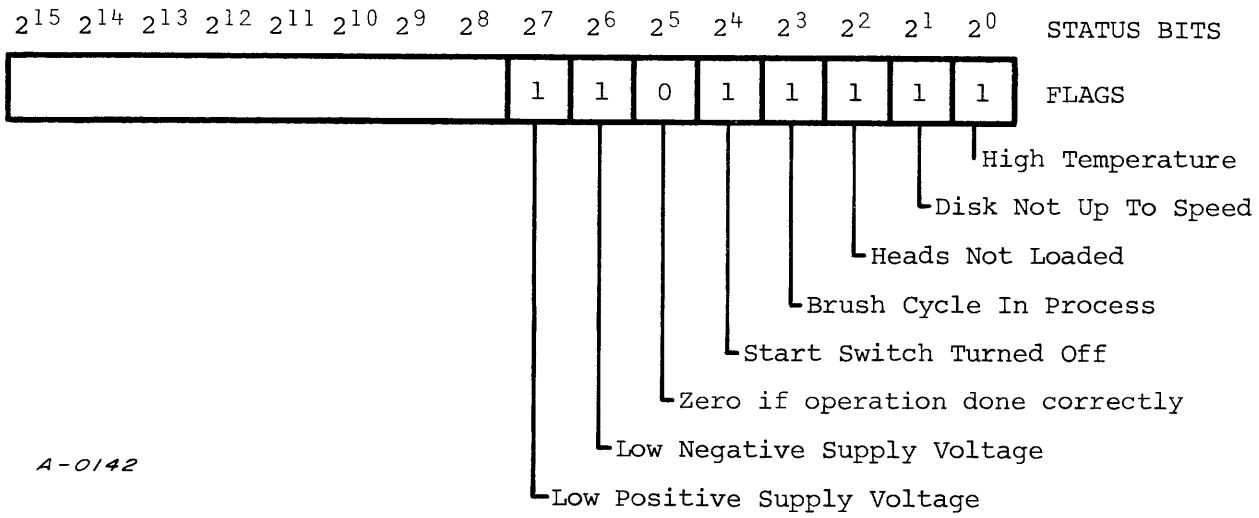


Figure 7-6. Interlock register status bits

Timing Notes

A function 4 select head group request can be followed immediately by another function to be done as soon as the new head group is active. For example, a read disk data function may be stacked behind the select head group function. If a function 1 status request is sent to the interface before the read disk data function begins, the function 1 command takes the place of the read disk data function, and the read disk data function is lost. In this case, the Lost Function error flag sets in the interface and will be available for the Read Error flags status request.

It is possible to issue a function 1 status request after a select cylinder function and before the select cylinder function has finished. But the status request should be issued immediately after the select cylinder function, to give plenty of time separation between the done flag of the status request and the done flag of the cylinder select. If the two done flags occur close together, the program may not be able to distinguish between them and handle the conditions incorrectly. It is best to avoid status request functions near the end of a head positioning sequence.

Table 7-5. Interlock status bits

Bit	Meaning
2^0	This bit is set to indicate that the disk drive cabinet is over the normal temperature range.
2^1	This bit is set to indicate that the disk surfaces are not up to speed.
2^2	This bit is set to indicate that the disk heads are not loaded on the disk surface.
2^3	The bit is set to indicate that the disk drive brush cycle is in process.
2^4	This bit is set to indicate that the disk drive start switch is turned off.
2^5	This bit is always zero.
2^6	This bit is set to indicate that the negative voltage supply for the disk drive is below normal voltage.
2^7	This bit is set to indicate that the positive voltage supply for the disk drive is below normal voltage.

DKA : 2 - READ DISK DATA

A function 2 request to the disk channel interface begins the process of reading a block of disk data into the I/O Memory at the beginning address specified by the interface A register. The disk sector number is specified by the accumulator content at the time of the function request. The head group number and cylinder number are the values which were last selected by the appropriate functions for that purpose.

The reading process begins by a hardware test for proper angular position of the disk surface. The sector number requested by the processor is compared with the sector number currently under the reading heads. If the disk is not in the proper position, the execution of the read function is delayed until the disk surface is properly positioned.

The interface anticipates the processor request for reading data when other function requests have been satisfied. This anticipation takes the form of reading each sector as the data appears under the reading heads with the expectation that the read request for that data will be forthcoming.

Data read from the disk surface is routed to the I/O Memory through two buffers, labeled A and B. Each buffer holds 512 words of I/O Processor data. Buffer A receives the first 512 words of disk data. Buffer B receives the second 512 words. The use of the two buffers alternates until the entire block of data has been processed.

The function 2 request clears the channel done flag and sets the channel busy flag. The sector number is captured from the accumulator and compared with the sector count. The currently anticipated reading process will be accepted if buffer A has not yet filled with the first sector of data from the disk surface. If the read process has proceeded beyond this point, or if the sector number is wrong, the current read process is aborted and the function request waits up to one disk revolution for sector coincidence.

The data in buffer A begins moving to the processor I/O Memory as the disk read circuits begin filling buffer B. The data moves to the I/O Memory in bursts of four words and normally empties buffer A before the disk read circuits have filled buffer B. The roles of the two buffers then reverse and data continues moving from the disk surface to the I/O Memory until the entire block has been processed.

The eight words of error correction data which follow the data block are read into a buffer as the last section of the data block is moving from the other buffer to the I/O Memory. Reading stops at this point for a check of the Fire code generators which have been summing the data as it was read from the disk surface. If all four generators are clear, the data read from the disk is correct. The channel done flag is set and the channel busy flag is cleared as the last word of data is entered in the I/O Memory.

DKA : 2 - Abnormal Conditions

An abnormal condition in the disk storage unit or in the reading or processing of the data is indicated to the I/O Processor by a terminating sequence which sets the channel done flag and leaves the channel busy flag set as well. The processor program can then analyze the error by appropriate function 1 requests for status information. Three types of error conditions cause this termination, as described below.

Recorded data error - While the recorded data is transferring from the DSU to the interface, the Fire code generators operate on the data as explained in "Fire Code Generator" later in this section. The 32-bit error correction code from each read head also passes through the Fire code generator for that head. The four new error correction codes generated should each be 32 zeros, if the data was stored and read correctly.

If any error correction code is non-zero, the recorded data error flag for that head is set in the fault register. There are four such flags for the four read channels. The I/O processor may use a special read mode to obtain the error correction code necessary to proceed with the correction software algorithm. The special read mode is described in the following "DKA : 2 - Special Modes" section.

Lost data error - The lost data flag is set in the fault register if the transfer of data from the buffers did not keep up with the reading of data from the disk surface. The processor must reread the sector in this case.

Lost Function - The lost function flag in the error status response is set if a function is received at the interface before a previous function has finished. The new function is lost, and should be requested again.

DKA : 2 - Special Modes

Special modes for the reading of disk data are requested by the processor through sector numbers larger than 40 octal. The low-order 5 bits of the requested sector number are translated for sector coincidence. The high-order bits are interpreted for special mode.

Format mode - Sectors 40_8 through 61_8 request that the reading process begin with the verification word and continue for a total of 1000 octal words. The following eight words are then interpreted as the error correction code. This read mode is used for maintenance only. In this case the error correction code was not generated by the Fire code generators because the data length during the write sequence was much longer than the requested length for this mode. This combination of long write and short read allows the maintenance routine to test for various failure modes in the Fire code generators.

Read correction code - Sectors 100_8 through 121_8 request that the reading process begin with the error correction code and continue for the eight words of that code. This mode is used when the data in the associated sector has been read incorrectly and the processor program wishes to do error correction.

Read early - Sectors 200_8 through 221_8 cause the reading heads to sample the data from the disk surface somewhat earlier than normal. This mode is used to recover data that cannot be read in a normal mode.

Read late - Sectors 400_8 through 421_8 cause the reading heads to sample the data from the disk surface somewhat later than normal. This mode is used to recover data that cannot be read in a normal code.

Mixed modes - The read early and read late selections described above may be used together with the other special modes by combining the high-order bit values in a logical sum.

Buffer Echo Mode

A special diagnostic feature allows writing a test data block into the A and B buffers and then reading out the test data. This can only be done after a master clear and before a DKA : 1 function. The master clear could be from a power on sequence or from a deadstart sequence. A DKA : 14 function to enter the starting address for the read to I/O Memory must precede the DMA : 2 read function. Refer to the buffer echo mode explanation given with the following DKA : 3 write function discussion.

DKA : 3 - WRITE DISK DATA

A function 3 request to the disk channel interface begins the process of writing a block of data from the I/O Memory onto the disk surface. The I/O Memory address for beginning the block of data is specified by the content of the interface A register. The disk sector number is specified by the accumulator content at the time of the function request. The head group number and cylinder number are the values which were last selected by the appropriate functions for that purpose. The function request sets the channel busy flag and clears the channel done flag.

The writing process begins by filling buffer A with data from the I/O Memory. When this buffer is full, the interface monitors the angular position of the disk surface for the proper position to begin writing data. This position is slightly past the prerecorded verification word for the requested sector. The write circuits are turned on when the disk is in the proper position and the data in buffer A is transmitted to the disk surface. At this time the interface begins filling buffer B with data from the I/O Memory. Buffer B should be filled before the disk writing circuits have emptied buffer A. The lost data flag in the fault register is set if this should not be the case. The disk writing circuits begin transmitting data from buffer B to the disk surface as soon as buffer A has been emptied. This process continues with the roles of buffers A and B alternating until the last of the data in the block has been loaded into buffer B. No further data is read from the I/O Memory and the interface waits for completion of the data transfer from buffer A to the disk surface.

The interface sets the channel done flag as soon as buffer A is emptied for the last portion of the disk data. The channel busy flag is cleared if no error has occurred in the writing process. The interface then continues transmitting data from buffer B to the disk surface. The I/O Processor is free at this point to issue another write function request and begin loading buffer A with data for another sector on this track.

The disk write circuits follow the last data from buffer B with eight words of error correction code. This data comes directly from the Fire code generators. The writing circuits are turned off at this time and the interface prepares to write data in the following sector if the I/O Processor has requested this function.

The busy flag is left set at the time the done flag is cleared if an error has occurred during the writing process. There are two possible error conditions, which are described in the following paragraphs.

Fire Code Generation

As data is transferred from the I/O Processor into the interface buffer, the interface generates the Fire code, or cyclic redundancy checkword (CRC). The Fire code generator uses the polynomial $x^{32} + x^{23} + x^{21} + x^{11} + x^{2} + 1$ to create one 32-bit checkword for each of the four DSU write heads. This polynomial allows for the correcting of data in a single error burst of 11 bits or less; and detects errors if there are two bursts, or bursts of more than 11 bits in error.

The checkword is generated by a 32-bit shifting register; the register is cleared before a data transfer is begun. The data bit going to the buffer is compared to bit 2^{31} of the shifting register. If the two bits are alike, bit 2^{0} of the register is cleared and the register is shifted left one position. If the bits are not alike, register bits 2^{1}, 2^{10}, 2^{20}, and 2^{22} are complemented, the register contents are then shifted left one position, and bit 2^{0} of the register is set to 1. Then, the next data bit is compared with register bit 2^{31} and the register contents are altered as previously.

Data bit 2^{63} is the first data bit to be used in generating the checkword for DSU head 3; 2^{62} is the first data bit used for DSU head 2, 2^{61} for head 1, and 2^{60} for head 0. Bits 2^{47}, 2^{31}, and 2^{15} are the other bits used first of their 16-bit channel words destined for DSU head 3.

The checkword forms continuously while the interface transfers data to the DSU. The 32-bit error correction code is appended to the data block.

Lost Data Error

The lost data flag is set if the data transfer from the I/O Memory has not kept up with the data transfer to the disk surface.

Lost Function Error

The status response lost function error flag is set if a function is received at the interface before the previous function is finished. The new function is lost. To recover, issue the new function again.

Format Mode

A special mode of operation is provided in the disk channel interface to prerecord the cylinder verification data on the disk surface. This mode is selected by a function 3 request with a sector number value of 40 through 61 octal. The disk control circuits translate the low-order 4 bits of the sector number in the normal way to select the sector for recording. The high-order bit causes the writing process to begin process to begin earlier than normal. This selection records a 4000_8 word block of data on the disk surface in an otherwise normal manner in such a way that the first word of this data block is properly positioned for the cylinder verification word. The remainder of the recording in this format mode is used for maintenance functions and is then erased by the recording of the normal sector data. Figure 7-7 shows the pattern written in each sector under format mode.

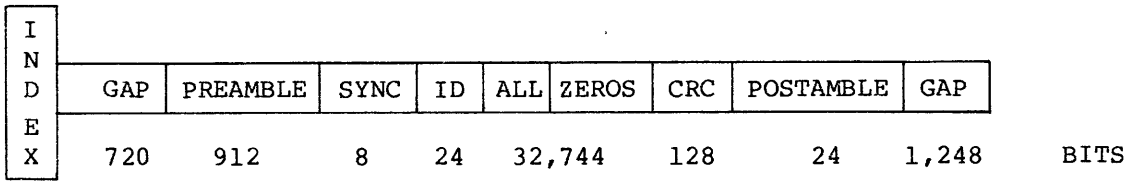


Figure 7-7. Format mode sector pattern

Buffer Echo Mode

A special diagnostic feature allows writing a test data block into buffers A and B and then reading out the test data. This can only be done after a master clear (from a power on sequence or a deadstart) and before a DKA : 1 function. The sequence is as follows:

1. Master clear
2. DKA : 14 enter I/O Memory starting address
3. DKA : 3 write data to buffers A and B
4. DKA : 14 enter I/O Memory starting address
5. DKA : 2 read data from buffers
6. Verify data
7. DKA : 1 function places controller in normal operation mode

DKA : 4 - SELECT HEAD GROUP

A function 4 request to the disk channel interface reserves the DSU and causes the head group selection circuits to select a head group. The new head group number is specified by the low-order 4 bits of the accumulator content at the time of the function request. This function may be requested at any time with respect to the execution of other disk channel functions. The 4-bit code for selection of the head group is captured in a special register in the disk channel interface. The action of switching the head group circuits requires about 5 microseconds and is delayed until the completion of any other function currently in process. It is possible, therefore, to select a new head group during a read or write sequence and continue the reading or writing from the last sector of one track to the first sector of a different track in the same disk cylinder, without missing a disk revolution.

If this function is used to reserve the DSU, any one of the DKA : 1 functions must still be issued before this DKA : 4 function.

This function request does not alter the condition of the channel busy or done flags. There is no channel interface response to this function.

DKA : 5 - SELECT CYLINDER

A function 5 request to the disk channel interface causes the disk read/record head assembly to move to a cylinder position. The cylinder number is specified by the 10 low-order bits of the accumulator content at the time of the function request.

The channel busy flag is set and the channel done flag is cleared by the function request. The servomechanism for positioning the head assembly then begins moving to the new cylinder position. This process takes from 15 milliseconds for adjacent cylinders to 80 milliseconds for maximum travel. A cylinder selection for the current cylinder requires a few microseconds for the process.

The interface monitors the cylinder positioning and when the read/record heads are on the newly requested cylinder, the data recorded on the selected track is read for verification of the cylinder. The data in the first verification word to pass under the read heads is captured and entered in the status response register. The channel busy flag is cleared to indicate completion of the requested function and the channel done flag is set. If the function cannot be completed because of an abnormal condition in the disk storage unit the channel done flag is set and the channel busy flag is left set.

It is possible to program the disk channel interface in such a way that the progress of the cylinder positioning can be monitored. This is done by aborting the normal sequence described above with a function 1 status request. This resets the busy flag and clears the done flag and begins the status response sequence. The cylinder positioning will continue but the verification process will not occur. The progress of the cylinder positioning can then be monitored by reading the difference register content from the disk storage unit. A verification can be programmed by repeating the cylinder selection.

DKA : 6 - CLEAR INTERRUPT ENABLE

A function 6 request to the disk channel interface clears the channel interrupt enable flag. This prevents interruption of the I/O Processor program and requires program monitoring of the channel done flag for proper sequencing of disk control functions.

DKA : 7 - SET INTERRUPT ENABLE

A function 7 request to the disk channel interface sets the channel interrupt enable flag. This causes an I/O interrupt request for this channel whenever the channel done flag is set. The interrupt enable flag function for the I/O Processor system.

DKA : 10 - READ LOCAL MEMORY ADDRESS

This function request reads the current value in the channel interface I/O Memory address register and enters this value in the accumulator. This function may be performed at any time with respect to a disk storage unit sequence.

DKA : 11 - READ STATUS RESPONSE

This function request reads the current content of the disk storage unit status response register and enters this value in the accumulator. This function request may be performed at any time with respect to the disk storage unit sequence. The value read will be the response from the last function which entered the status response register.

DKA : 14 - ENTER LOCAL MEMORY ADDRESS

This function enters the current accumulator content in the channel interface I/O Memory address register. The channel busy and channel done flags are not altered in this process.

DKA : 15 - STATUS RESPONSE REGISTER DIAGNOSTIC

This function can be used to verify the operation of the status response register. The function transfers a test value from the accumulator to the status response register, overwriting the current status. A DKA : 11 read status response function immediately following this diagnostic function will return the test value to the accumulator for verification. The channel busy and done flags are not affected by the DKA : 15 function.

CONSOLE DISPLAY CHANNEL

The I/O Processor has provision for a number of I/O channels connecting to operator display consoles. Each display is assigned to a separate channel and all may operate independently. Data is transmitted serially from a channel interface register to the display device.

This channel has a 7-bit interface register which receives data from the I/O Processor accumulator and transmits this data serially to the display device. Function requests for this channel 1 are described below.

TOA : 0§ Clear the channel busy and channel done flags.

TOA : 6§§ Clear the channel interrupt enable flag. The channel busy and channel done flags are not altered in this process.

TOA : 7§§ Set the channel interrupt enable flag. The channel busy and channel done flags are not altered in this process.

TOA : 14§ Enter the low-order 7 bits of the accumulator content in the channel interface register. It sets the channel busy flag and clear the channel done flag. It begins the transmission of the data from the interface register to the display device and clears the channel busy flag and set the channel done flag when the transmission has been completed.

§ Valid channel busy and channel done flags cannot be read until 1 CP after this function is issued.

§§ Allow 1 CP before checking the interrupt channel number (IOR : 10)

CONSOLE KEYBOARD CHANNEL

The I/O Processor has provision for a number of I/O channels connecting to operator console keyboards. Each keyboard is assigned to a separate channel and all may operate independently. Data is transmitted serially from the keyboard to a channel interface register. The channel busy flag is set by the channel hardware at the beginning of the transmission. The channel busy flag is cleared and the channel done flag is set when the data has been assembled in the interface register.

This channel has a 7-bit interface register which assembles the data for a character associated with a key depression. This data may then be read into the I/O Processor accumulator.

TIA : 0§ Clear the channel done flag.

TIA : 6§§ Clear the channel interrupt enable flag. The channel busy and channel done flags are not altered in this process.

TIA : 7§§ Set the channel interrupt enable flag. The channel busy and channel done flags are not altered in this process.

TIA : 10§ Read the contents of the channel interface register into the low-order 7-bit positions in the accumulator and clear the high-order bits. It sets the channel busy flag and clears the channel done flag. It transfers the data to the accumulator, then clears the channel busy flag and sets the channel done flag.

---

§ Valid channel busy and channel done flags cannot be read until 1 CP after this function is issued.

§§ Allow 1 CP before checking the interrupt channel number (IOR : 10)

PERIPHERAL EXPANDER CHANNEL

Depending on its function, an I/O Processor may have a channel connected to a peripheral expander. The peripheral expander can contain 16 controllers for peripheral devices. Only one controller can be active at one time, but that controller may be servicing more than one peripheral device. Each peripheral unit is assigned a device address and is individually selectable. The functions for the peripheral expander are listed below.

EXB : 0§	Idle channel
EXB : 1§	DIA data input from A register
EXB : 2§	DIB data input from B register
EXB : 3§	DIC data input from C register
EXB : 4§	Read busy/done, interrupt number
EXB : 5	Load device address
EXB : 6§§	MSKO mask out
EXB : 7§§	Set interrupt mode
EXB : 10	Read data bus status
EXB : 11	Read status 1
EXB : 13	Read status 2
EXB : 14§	DOA data output to A register
EXB : 15§	DOB data output to B register
EXB : 16§	DOC data output to C register
EXB : 17§	Send control

INTERFACE REGISTERS

Each peripheral controller has three interface registers: A, B, and C. These registers are used for control and data communication between the peripheral device and the I/O Processor. The specific uses of the registers are defined by the peripheral device controller.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

CHANNEL ASSIGNMENTS

All peripheral devices handled by the peripheral expander share the same channel number. Device addresses select among the peripheral units.

EXB : 0 - IDLE CHANNEL

This instruction clears the peripheral expander channel busy and done flags, clears the peripheral expander interrupt enable flag, and clears the interface DMA (direct memory access) enable flag. The peripheral expander channel is then inactive and DMA references via the data channel will not be allowed.

EXB : 1 - DIA

This instruction requests the A input register contents from the selected peripheral controller in the peripheral expander. Peripheral expander channel busy sets and done clears, both in the clock period following issue. Since this command requires the use of the data bus, it is delayed by the function delay time (minimum 1 microsecond). When it completes, the peripheral expander channel done flag sets and the busy flag clears. At this time, the I/O Processor may follow this instruction with an EXB : 10 to load the A input register information into the I/O Processor accumulator.

EXB : 2 - DIB

This instruction performs exactly as the EXB : 1 instruction, except that the peripheral controller B input register is sampled.

EXB : 3 - DIC

This instruction performs exactly the same as the EXB : 1 instruction, except that the peripheral controller C input register is sampled.

EXB : 4 - READ BUSY/DONE, INTERRUPT NUMBER

This instruction serves two purposes. The first purpose is to return the specified peripheral controller busy and done flags to the interface. The second is to determine which peripheral controller has the highest priority interrupt, its done flag set, and its interrupts enabled.

This is a delayed function because it uses the bus data lines from the peripheral expander. The peripheral expander channel busy flag sets and done flag clears 1 CP after instruction issue. Upon completion, the peripheral expander channel busy flag is cleared and the done flag is set. At this time, an EXB : 11 instruction may be issued to load the peripheral controller busy flags, done flags, and the present interrupt device address.

EXB : 5 - LOAD DEVICE ADDRESS

This instruction loads the bits 2^0 - 2^5 of the I/O Processor accumulator into the interface device address register. The interface device address register is used to hold the device address of a peripheral controller to which a delayed function is being sent. A delayed function is any function which requires the use of the bus data lines in the expander, thus requiring time to complete. Functions that are considered delayed functions are the EXB : 1, 2, 3, 4, 6, 14, 15, 16, and 17 functions. The address in the interface device address register must not be changed at any time during the execution of a delayed function. This instruction does not change the peripheral expander channel done and busy flags. The device address is loaded into the interface register in the I/O Processor clock period following instruction issue.

EXB : 6 MSKO MASK OUT

This instruction sends the present contents of the I/O Processor accumulator to the peripheral expander where the 16-bit word acts as a mask to disable interrupts from specific peripheral controllers. Every device controller in the peripheral expander has a mask bit. The specific mask bits for some probable peripheral devices are given in table 7-6. The mask bits are subject to change, so the latest documentation for the peripheral interface should be consulted. Three Cray Research assignments are shown on the table. If the mask bit is set, interrupts from the corresponding peripheral controller are disabled. The peripheral expander channel busy flag is set and the done flag is cleared upon initiation of this instructon. Because the mask must be sent throughout the peripheral expander via the bus data lines, this instruction is a delayed instruction. When it does complete, the peripheral expander channel busy flag clears and the done flag sets.

Issuing this instruction clears the interrupt flag. If an interrupt condition is still present 400 nanoseconds after completion, the interrupt flag will set again.

Table 7-6. Peripheral device mask bits for interrupt disabling

Octal Device Code	Mask bit	Device
11/51§	2^0	Teletype output
11/50§	2^1	Teletype input
30/70§		Asynchronous hardware multiplexer
70		Synchronous Line Adapter
14/54§	2^2	Real-time clock option
31,32/71,72§		IBM 360/370 interface
06/46§	2^3	Multiprocessor adapter transmitter
07/47§		Multiprocessor adapter receiver
15/55§		Incremental plotter
17/57§		Card Reader CR0 (CRI modification)
44		Modem control for multiline asynchronous controller
34,35/74,75§	2^4	Multiline asynchronous controller
16/56§	2^5	Magnetic tape MT0 (CRI)
22/62§		Cassette tape
34/74§		
20/60§	2^6	Fixed head disk
21/61§	2^7	Analog/Digital converter
40		Interprocessor bus full-duplex unit
41		Interprocessor bus full-duplex unit
40§§		Synchronous communication receiver
41§§§		Synchronous communication transmitter
33/73	2^8	Moving head disk
42		Digital I/O
43	2^9	Line Printer PR0 (CRI modification)
64,65,66/74	2^{10}	Floating-point (NOVA only)
76,76§		
	2^{11}	Unassigned
	2^{12}	Unassigned
	2^{13}	Unassigned
	2^{14}	Unassigned
	2^{15}	Data communications multiplexer

§ First device code/second device code
§§ May be set up with any unused even device code greater than 40_8
§§§ May be set up with any unused odd device code greater than 41_8

EXB : 7 - SET INTERRUPT MODE

This instruction is used to enable or disable interrupts for the peripheral expander I/O channel and for the peripheral controllers. The accumulator value present when this instruction issues determines what types of interrupts are honored. If a bit is set, the interrupts will be enabled as follows:

Accumulator bit 2^0 - Enables interrupts from the I/O channel to the I/O Processor

Accumulator bit 2^1 - Enables interrupts from the I/O controllers to the I/O Processor

The I/O channel busy and done flags are not affected by this instruction.

EXB : 10 - READ DATA BUS STATUS

This instruction reads the data on the data bus into the I/O Processor accumulator. This is normally used to bring back the data received from the bus by the EXB : 1, 2, or 3 input instructions. The peripheral expander I/O channel busy and done flags are not effected by this instruction. The data will stay valid until another EXB : 1, 2, or 3 function is issued.

EXB : 11 - READ STATUS 1

This instruction returns to the I/O Processor the status of the peripheral expander I/O channel and of the peripheral controller. Channel busy and done flags are not affected by this instruction. The status bits are assigned as shown in table 7-7.

Table 7-7. Read status 1 bit assignments

Bit	Meaning
2^0	Interrupting device code bit 2^0
2^1	Interrupting device code bit 2^1
2^2	Interrupting device code bit 2^2
2^3	Interrupting device code bit 2^3
2^4	Interrupting device code bit 2^4
2^5	Interrupting device code bit 2^5
2^6	Unassigned
2^7	Direct memory access enabled
2^8	Expander I/O channel interrupts enabled
2^9	Controller interrupts enabled
2^{10}	Function active - delayed function executing
2^{11}	Expander I/O channel busy flag
2^{12}	Expander I/O channel done flag
2^{13}	INTR - interrupt request from peripheral interface
2^{14}	SELB - select busy flag of addressed device
2^{15}	SELD - select done flag of addressed device

EXB : 13 - READ STATUS 2

This instruction returns to the I/O Processor the status of the peripheral expander I/O channel and of the peripheral controller. Channel busy and done flags are not affected by this instruction. The instruction is similar to EXB : 11, except that the contents of the device address register in the interface is returned instead of the interrupt device code number. Status bit assignments are shown in table 7-8.

Table 7-8. Read status 2 bit assignments

Bit	Meaning
2^0	Device address bit 2^0
2^1	Device address bit 2^1
2^2	Device address bit 2^2
2^3	Device address bit 2^3
2^4	Device address bit 2^4
2^5	Device address bit 2^5
2^6	Unassigned
2^7	Direct memory access enabled
2^8	Expander I/O channel interrupts enabled
2^9	Controller interrupts enabled
2^{10}	Function active - delayed function executing
2^{11}	Expander I/O channel busy flag
2^{12}	Expander I/O channel done flag
2^{13}	INTR - interrupt request from peripheral interface
2^{14}	SELB - select busy flag of addressed device
2^{15}	SELD - select done flag of addressed device

EXB : 14 - DOA (DATA OUT A)

This instruction sends the contents of the I/O Processor accumulator to the selected peripheral controller in the peripheral expander. The data goes into the A register of the peripheral controller whose device address is currently in the device address register. This command causes the peripheral expander I/O channel busy flag to set and the done flag to clear. This command which requires the use of the data bus in the peripheral expander and is a delayed function. When it does complete, the peripheral expander I/O channel done flag sets and busy clears.

EXB : 15 - DOB (DATA OUT B)

This instruction sends the contents of the I/O Processor accumulator to the selected peripheral controller in the peripheral expander. The data goes into the B register of the addressed device controller. The instruction operation is otherwise the same as the EXB : 14 DIA instruction.

EXB : 16 - DOC (DATA OUT C)

This instruction sends the I/O Processor accumulator contents to the C register of the selected peripheral controller in the peripheral expander. Otherwise, the operation is the same as the EXB : 14 DOA instruction operation.

EXB : 17 - SEND CONTROL

The peripheral controllers use four control signals in the peripheral expander: I/O Reset, Pulse, Clear, and Start. Each controller may have different uses for these signals. This instruction sends the control signal selected by the bits in the accumulator at instruction issue. The control signals are delayed instructions. The peripheral expander I/O channel busy flag sets at issue time and the done flag clears. Then when the command has completed, the channel done flag sets and the busy flag clears. The accumulator bits set control signals as shown in table 7-9.

Table 7-9. Accumulator bit control signals

Bit	Function
2^0	Start (S)
2^1	Clear (C)
2^2	Pulse (P)
2^3	I/O reset (IORST)

DELAYED FUNCTIONS

Several functions require the data bus in the peripheral expander. These need extra time to complete, and may have to wait for the data bus to be free. The completion of a delayed function is shown by setting the peripheral expander I/O channel done flag. The done flag for the preceding delayed function must be received at the I/O Processor before sending another delayed function.

Those functions that are considered delayed functions are:

EXB : 1	DIA
EXB : 2	DIB
EXB : 3	DIC
EXB : 4	Read busy/done, interrupt number
EXB : 6	MSKO
EXB : 14	DOA
EXB : 15	DOB
EXB : 16	DOC
EXB : 17	Send control

TRANSFER SPEEDS

The peripheral expander interface is capable of sustaining a transfer rate of 16 megabits per second from the I/O Processor to the expander chassis. In the reverse direction, it can sustain a speed of approximately 14.5 megabits per second. This is achieved by using the data channel mode which transfers a 16-bit parcel every microsecond. The programmed I/O mode can also reach these speeds.

CHANNEL FOR INPUT FROM CRAY-1 CHANNEL

The I/O Processor may have one or more channels dedicated to receiving data from a CRAY-1 I/O channel. Data is transferred in block mode directly into I/O Processor I/O Memory. The functions are listed below.

CIA : 0§	Clear channel
CIA : 1§	Enter I/O Memory address, start transfer
CIA : 2	Enter parcel count
CIA : 3	Clear channel parity error flags
CIA : 4	Clear ready waiting flag
CIA : 6§§	Clear interrupt enable flag
CIA : 7§§	Set interrupt enable flag
CIA : 10	Read I/O Memory address
CIA : 11	Read ready waiting/error flags

I/O MEMORY ADDRESS REGISTER

The I/O Memory address register contains the I/O Memory address for the next I/O Memory reference. It is loaded with the I/O Memory starting address at the initiation of the input transfer. The low-order 2 bits of starting address are forced to 0 when loaded into the register. This is done because the memory references are done in bursts of four parcels, and upon completion of a reference, the register address is increased by 4. If, for some reason, the input transfer stops on some boundary other than four, the final memory reference stores undefined parcels. The number of defined parcels can be determined by reading the final memory address, which is equal to the address of the last defined parcel, plus 1.

CIA : 0 - CLEAR CHANNEL

This function clears the interface channel busy and done flags and aborts any transfer in progress. The new states of busy and done are not valid until the second clock period following this function. Wait one clock period after issuing this function before sampling these flags.

---

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

CIA : 1 - ENTER I/O MEMORY ADDRESS

This function enters the accumulator content into the 16-bit I/O Memory address register (forcing the 2 low-order bits to 0) and starts an input transfer from the Central Processing Unit. The busy flag is set and the done flag is cleared. Upon receipt of four parcels, a memory reference is made. Then another four parcels are received and stored. This continues until the parcel count is 0 or until a disconnect is received from the CRAY-1 I/O channel. At that time, the done flag is set and the busy flag is cleared.

CIA : 2 - ENTER PARCEL COUNT

This function stores the accumulator content into the interface parcel count register. This value is a positive count of the number of parcels to be transferred. The status of the busy and done flags are not affected.

CIA : 3 - CLEAR CHANNEL PARITY ERROR FLAGS

Four parity bits protect the 16-bit parcels on the CPU channel. A parity error in a 4-bit group causes one parity error flag to set. This funtion clears all four parity error flags. A parity error does not affect the states of the channel busy and done flags. Similarly, issuing this function does not affect the states of the channel busy and done flags.

CIA : 4 - CLEAR READY WAITING

If the I/O Processor input channel is inactive and a ready pulse is received from the CPU, the channel sets a ready waiting flag. If, in order to resynchronize the transfer, this ready must be discarded, this function will clear the ready waiting flag. If the ready signal is not discarded and the input channel is started, the data on the lines will be sampled as the first parcel of the transfer. The channel busy and done flags are not affected.

CIA : 6 - CLEAR INTERRUPT ENABLE FLAG

This function disables the interface from interrupting the I/O Processor. The channel may still be monitored via the busy and done flag status. The channel busy and done flags are not affected by this function.

CIA : 7 - SET INTERRUPT ENABLE FLAG

This enables interrupts on the interface. The channel will then interrupt whenever the done flag sets. The channel busy and done flags are unaffected by this function.

CIA : 10 - READ MEMORY ADDRESS

This function transfers the content of the interface I/O Memory address register into the accumulator. The address is one greater than the address of the last parcel stored. Since a memory reference is four parcels of data, there will be undefined parcels of data written into memory if the transfer length is not a multiple of four. The channel busy and done flags are unaffected by this function.

CIA : 11 - READ READY WAITING/ERROR FLAGS

This function reads into the accumulator the content of the interface status register. The status bit assignment is listed in table 7-10.

Table 7-10. Ready waiting/error flags

Bit	Meaning
2^0	Channel parity error flag for bits 2^0-2^3
2^1	Channel parity error flag for bits 2^4-2^7
2^2	Channel parity error flag for bits 2^8-2^{11}
2^3	Channel parity error flag for bits 2^{12}-2^{15}
2^{15}	Ready waiting flag

CHANNEL FOR OUTPUT TO CRAY-1 CHANNEL

The I/O Processor may have one or more channels for sending data to a CRAY-1 input channel. Data is transferred in block mode directly from I/O Memory. The functions are listed below.

COA : 0§	Clear channel
COA : 1§	Enter I/O Memory address, start transfer
COA : 2	Enter parcel count
COA : 3	Clear error flag
COA : 4	Set/clear external control signals
COA : 6§§	Clear interrupt enable flag
COA : 7§§	Set interrupt enable flag
COA : 10	Read I/O Memory address
COA : 11	Read error flags

I/O MEMORY ADDRESS REGISTER

The I/O Memory address register contains the I/O Memory address for the next I/O Memory reference. It is loaded with the I/O Memory starting address at the initiation of the output transfer. The 2 low-order bits of the starting address are forced to 0 when entered in the register. This is done because the memory references are in bursts of four parcels, and upon completion of a memory reference, the register address is increased by 4. Upon completion of the output, the I/O Memory address in the register is one greater than the address of the first parcel sent.

COA : 0 - CLEAR CHANNEL

This function clears the channel busy and done flags and aborts any transfer. The busy and done flags are not valid for sampling until the second clock period following completion of this function.

---

§ Allow 1 CP before checking busy and done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

COA : 1 - ENTER I/O MEMORY ADDRESS

This function loads the current contents of the accumulator into the 16-bit I/O Memory address register (forcing the 2 low-order bits to 0) and starts the transfer to the CPU. The busy flag sets and the done flag clears. The interface makes a memory reference starting at the address contained in the I/O Memory address register, and outputs four parcels of data. Then another reference reads out another burst of four parcels. When the parcel count stored in the interface register has been reached, the transfer stops. The done flag sets and the busy flag clears.

COA : 2 - ENTER PARCEL COUNT

This enters the accumulator content into the interface parcel count register. This value is a positive count of the number of parcels to be transferred. The channel busy and done flags are not affected by this function.

COA : 3 - CLEAR ERROR FLAG

This command is used to clear the sequence error flag. The sequence error flag sets when there is a resume signal received from the CPU and the interface is not busy or there is an I/O Memory reference in progress. If the interface interrupt enable flag is set, the sequence error flag will cause an interrupt. The sequence error has no affect on the channel busy and done flags. The channel busy and done flags are not affected by this function.

COA : 4 - SET/CLEAR EXTERNAL CONTROL SIGNALS

This function sends control signals to the CPU. The signal selection is governed by set bits in the accumulator as shown in table 7-11.

Hold disconnect is used to stop the automatic disconnect that is sent at the end of a transfer.

Write disconnect sends a disconnect to the CPU with no data transferred.

Error channel resume is reserved for maintenance use with configurations without an error logging channel. RTC interrupt is reserved for use on systems where a programmable clock is not present.

The accumulator signal select bits are held in an interface register until they are altered by another COA : 4 function. The channel busy and done flags are not affected by this command.

Table 7-11. External control signal bits

Bit	Control Signal
2^8	Write disconnect
2^9	Hold disconnect
2^{10}	Unused
2^{11}	Dead dump
2^{12}	RTC interrupt
2^{13}	Error channel resume
2^{14}	I/O master clear
2^{15}	CPU master clear

COA : 6 - CLEAR INTERRUPT ENABLE FLAG

This function clears the channel interrupt enable flag to prevent the interface from interrupting the I/O Processor. The channel may be monitored via the busy and done flags and via the status information returned by the COA : 11 command.

COA : 7 - SET INTERRUPT ENABLE FLAG

This function sets the interrupt enable flag to allow interrupting the I/O Processor. With interrupts enabled, the interface will interrupt whenever the done flag sets or whenever a sequence error occurs. This function does not affect the busy and done flags.

COA : 10 - READ I/O MEMORY ADDRESS

This command enters the content of the I/O Memory address register into the accumulator. The address entered will be one greater than the address of the last parcel transferred from I/O Memory. The channel busy and done flags are not affected by this command.

COA : 11 - READ ERROR FLAGS

This function reads into the accumulator the status of the interface. The bit significances are listed in table 7-12.

Table 7-12. Error flags

Bit	Meaning
2^0	4-bit channel data bit 2^0
2^1	4-bit channel data bit 2^1
2^2	4-bit channel data bit 2^2
2^3	4-bit channel data bit 2^3
2^{15}	Sequence error

The channel busy and done flags are not affected by this command. The 4-bit channel is reserved for maintenance use with configurations without an error logging channel.

MEMORY CHANNEL

A Memory Channel (1 standard and 1 optional) is used for transferring data between the CPU Central Memory and an IOP I/O Memory. A Memory Channel consists of an input channel that carries data from Central Memory to I/O Memory, and an output channel that passes data from the I/O Memory to the Central Memory. Each channel carries 64 bits of data in parallel, along with 8 check bits. A separate 12-bit channel is provided for both the input channel and the output channel for carrying address information. A data rate of approximately 800 million bits per second is possible on either the input channel or the output channel.

The input channel and the output channel are more complex than the normal CRAY-1 channels. There are more control signals provided, and more data protection information. The signals and their timing are described in this section, followed by an explanation of the IOP instructions that control a Memory Channel.

Figure 7-8 shows the Memory Channel signals and the directions of the signals. First the input channel signals are described, then the output channel signals are described.

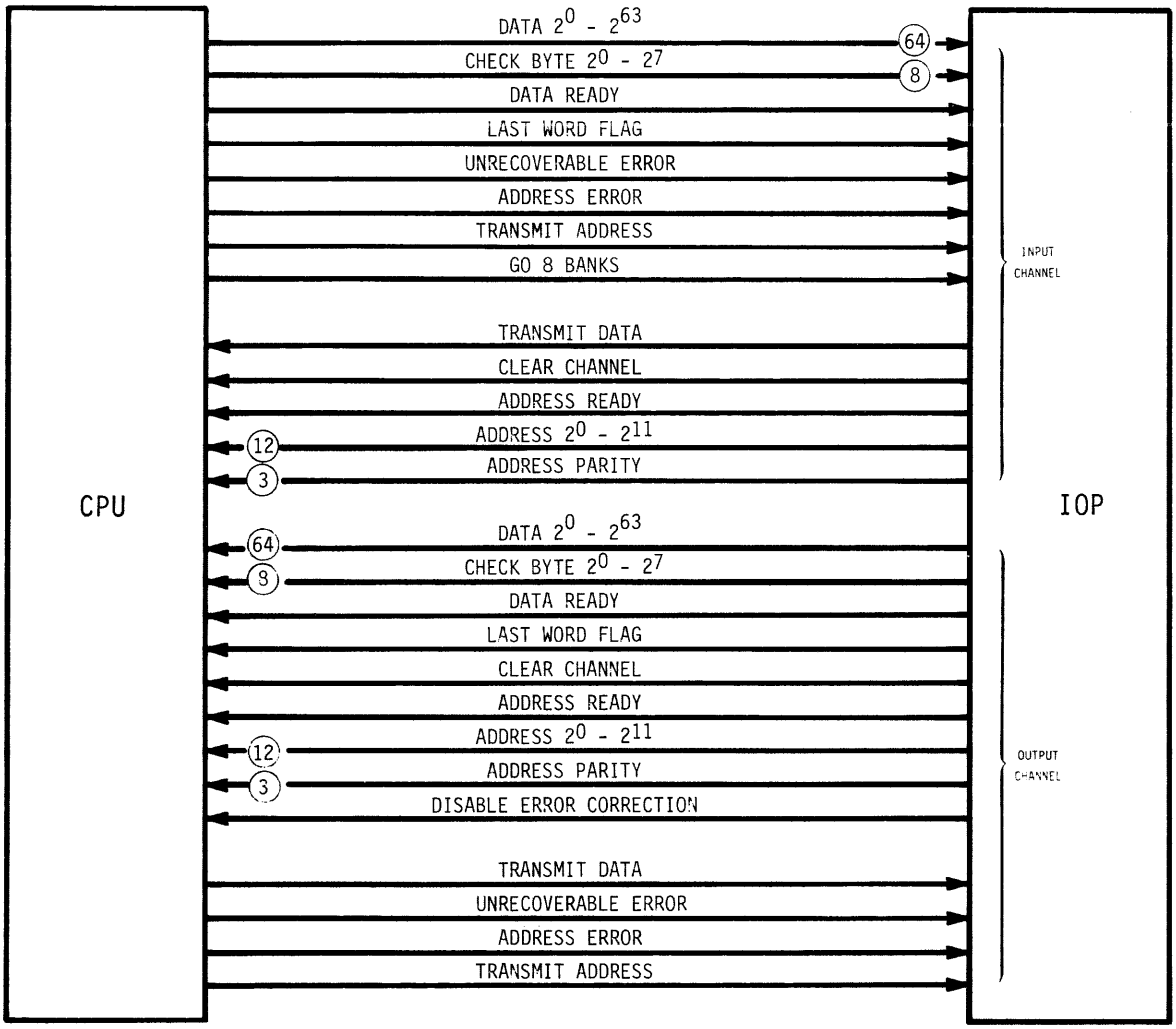


Figure 7-8. Memory Channel signals

Central Processing Unit to I/O Processor Input Channel

Data (to IOP) - The data is passed in 64-bit words over 64 lines. Data is valid 2 CPs after the leading edge of Data Ready, and stays valid for 6 CPs.

Check Byte (to IOP) - The Check Byte is the 8 error check bits read out of the CPU memory. It is passed to the I/O Processor to verify the accuracy of the received Data. The Check Byte has the same timing and valid points as the Data Signal.

Data Ready (to IOP) - This signal indicates data is valid on the cable. The leading edge of Data Ready preceded the Data signals by 2 CPs and is true for 1 CP. The minimum period for Data Ready signals is 6 CPs.

Last Word Flag (to IOP) - This signal indicates the last data word of the complete transfer is on the Data lines. This also means the CPU word count is 0. This signal has the same timing and duration as the Data signals.

Unrecoverable Error (to IOP) - This signal indicates an unrecoverable error occurred in the transfer at the CPU. The channel will go inactive and remain so until a Clear Channel signal is sent to the CPU from the IOP. Unrecoverable error conditions include:

1. Address parity error,

2. Transmit Data signal received at the CPU and less than 3 Address Ready signals were received,

3. More than 3 Address Ready signals received at the CPU,

4. Uncorrectable data error from Central Memory.

The unrecoverable error signal continues until a Clear Channel is sent.

Address Error (to IOP) - The Address Error signal indicates the unrecoverable error was an address error, from conditions 1, 2 or 3 of the error conditions listed previously in "Unrecoverable Error". The Address Error signal should be sampled when the leading edge of the Unrecoverable Error Signal appears.

Transmit Address (to IOP) - This signal indicates the CPU side is inactive and that a transfer can be initiated by sending an address on the Address line. Transmit Address returns to the cleared state after three Address Ready signals, and will not set again until after the last data word has been transferred to the IOP Processor. This signal also clears if there is an error condition on the CPU side of the channel and will not set again until a Clear Channel is sent.

Go 8 Banks (to IOP) - This signal indicates the Central Memory is operating in the 8-bank mode. All transmitted block sizes should be reduced to eight words.

Transmit Data (to CPU) - This signal indicates the I/O Processor can accept a block of data from the CPU. This signal remains set until the first Data Ready signal is received for that data block. Transmit Data is sampled at the CPU at the beginning of each data block. If Transmit Data is set, the 16-word data block will be transmitted. If this signal is cleared at that time, no data will be transmitted until the Transmit Data signal sets. The Transmit Data signal is clear when the channel is inactive and will not set until at least 100 nanoseconds after the third Address Ready signal trailing edge.

NOTE

The number of 64-bit data words sent in the first block equals 16 minus the number specified by bits 2^0 and 2^1 of the CPU starting address. This adjusts the blocks to 16-bank boundaries so that the next block begins with section 0. If the Go 8 Bank signal is set, the number of data words in the first block is 8 minus the address 2^0 value.

Clear Channel (to CPU) - This signal clears all error conditions in the channel. It remains set a minimum of 100 nanoseconds or until the Unrecoverable Error signal goes false. An Address Ready signal must lag the trailing edge of the Clear Channel signal by at least 100 nanoseconds.

Address Ready (to CPU) - This signal indicates the Address lines will soon have valid address data. The leading edge of this signal leads the address information by 25 nanoseconds and stays set for 12.5 nanoseconds. The minimum period for Address Ready signals is 75 nanoseconds, leading edge to leading edge. Three Address Ready signals are used to initiate a transfer. The first accompanies bits 2^{10} through 2^{21} of the beginning central memory address on the Address lines. The second Address Ready accompanies bits 2^0 - 2^9 of the beginning Central Memory address and bits 2^{13} - 2^{12} of the transfer word count. The third address Ready accompanies bits 2^0 - 2^{11} of the transfer word count. See figure 7-9.

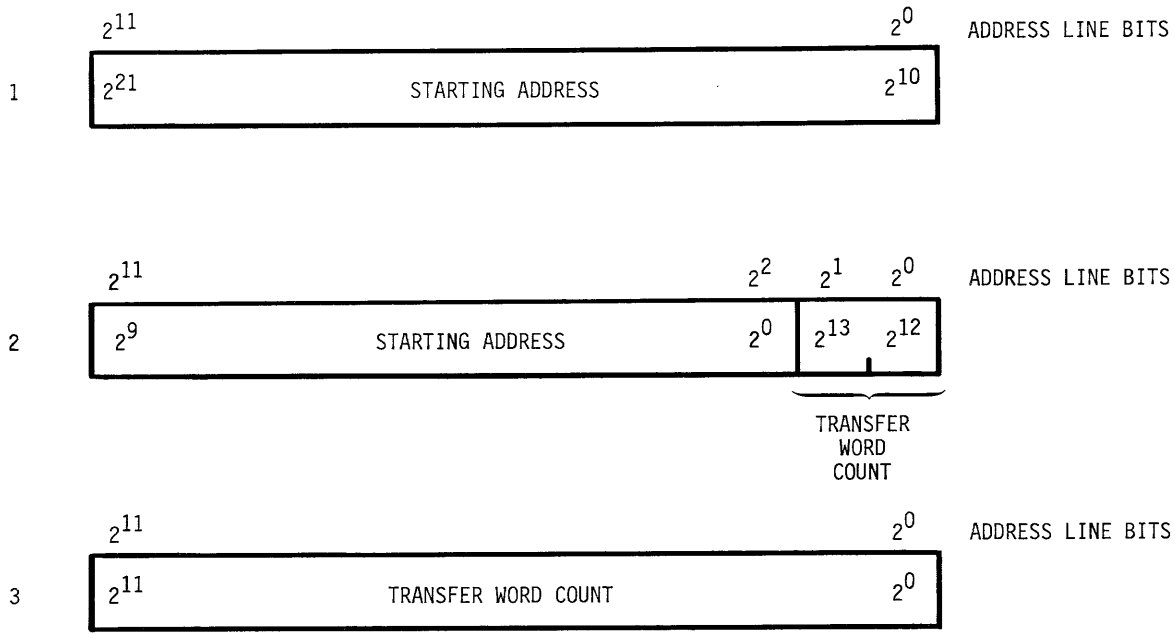


Figure 7-9. Address and word count formats

Address (to CPU) - The 12 Address lines carry the Central Memory starting address and the transfer word count. The information carried on the Address lines for each of the three address transfers is shown in figure 7-9. The information is valid for 75 nanoseconds. The Address signals lag the leading edge of Address Ready by 25 nanoseconds.

Address Parity (to CPU) - These signals provide odd parity for each group of 4 address bits as follows:

Address Parity Bit	Address Information Bits
2^0	$2^0 - 2^3$
2^1	$2^4 - 2^7$
2^2	$2^8 - 2^{11}$

The Address Parity signals have the same timing as Address signals. They are valid for 75 nanoseconds.

I/O Processor to Central Processing Unit Output Channel

Data (to CPU) - Data passes in 64-bit words over 64 lines. Data is valid 25 nanoseconds after the Data Ready signal leading edge and Data stays valid for 75 nanoseconds.

Check Byte (to CPU) - Check Byte is the 8 error check bits generated by the output side of the channel. The error code generation methods used are the same on each side of the channel. The Check Byte is passed to the CPU with the same timing and valid points as the Data signals.

Data Ready (to CPU) - This signal indicates data is valid on the cable. The leading edge of Data Ready precedes the Data signals by 25 nanoseconds and Data Ready is set for 12.5 nanoseconds. The minimum period for Data Ready signals is 75 nanoseconds.

Last Word Flag (to CPU) - This signal indicates the last data word of the complete transfer on the Data lines. This also means the I/O Processor word count is 0. This signal has the same timing and duration as the Data signals.

Clear Channel (to CPU) - This signal clears all error conditions in the channel. It remains set a minimum of 100 nanoseconds or until the Unrecoverable Error signal clears. An Address Ready signal must lag the trailing edge of the Clear Channel signal by at least 100 nanoseconds.

Address Ready (to CPU) - This signal indicates information will soon be valid on the Address lines. The leading edge of Address Ready leads the valid address information by 25 nanoseconds and is set for 12.5 nanoseconds. The minimum period for sequential Address Ready signals is 75 nanoseconds. The Address Ready signal is only sent when the Transmit Address signal is set. Three Address Ready signals are used to initiate a data block transfer. The first Address Ready signal accompanies Address information which is bits 2^{10} - 2^{21} of the starting address in Central Memory. The second Address Ready indicates the Address lines will soon contain bits 2^{0} - 2^{9} of the Central Memory starting address and bits 2^{12} - 2^{13} of the transfer word count. The third address information transmission preceded by an Address Ready, contains the 2^{0} - 2^{11} bits of the transfer word count. See preceding figure 7-9.

Address (to CPU) - The 12 Address lines carry the Central Memory starting address and the transfer word count. The Address signals lag the leading edge of Address Ready by 25 nanoseconds, and then stay set for 75 nanoseconds. The formats of the information carried in each of three address information transfers are shown in the preceding figure 7-9.

Address Parity (to CPU) - These signals provide odd parity for each group of four Address signals as follows:

Address Parity Bit	Address Information Bits
2^0	$2^0 - 2^3$
2^1	$2^4 - 2^7$
2^2	$2^8 - 2^{11}$

The Address Parity signals have the same timing as Address Signals. They are valid for 75 nanoseconds.

Disable Error Correction (to CPU) - This signal is used for diagnostic purposes only. It disables the Single Error Correction, Double Error Detection (SECDED) circuitry used on the data channel.

Transmit Data (to IOP) - This signal indicates the CPU can accept a block of data from the I/O Processor. This signal remains set until the first data ready signal is received for that data block. Transmit data is sampled at the beginning of each data block. If Transmit Data is set, the 16-word data block will be transmitted. If this signal is cleared when sampled, no data will be sent until Transmit Data does go set.

NOTE

The number of 64-bit words sent in the first block of a transfer equals the number of banks (16 or 8), minus the number specified by bits 2^0 and 2^1 (16 banks) or by bit 2^0 (8 banks) of the Central Memory starting address. This adjusts the blocks to 16-bank or 8-bank boundaries so that the next block begins with bank 0.

Unrecoverable Error (to IOP) - This signal indicates an unrecoverable error occurred in the transfer at the CPU side. The channel will go inactive and remain so until a Clear Channel is sent to the CP. Unrecoverable Error conditions include:

1. Address Parity Error,
2. Data Ready received and less than three Address Ready signals,
3. More than three Address Readies received,
4. Uncorrectable data error on channel,
5. Transfer word count = 0 and no Last Word Flag,
6. Transfer word count ≠ 0 and Last Word Flag.

The Unrecoverable Error signal remains set until a Clear Channel signal is sent.

Address Error (to IOP) - The Address Error signal indicates the Unrecoverable Error was an address error, from conditions 1, 2, or 3 of the preceding Unrecoverable Error list. The Address Error signal should be sampled when the leading edge of Unrecoverable Error appears.

Transmit Address (to CPU) - This signal indicates the CPU side is inactive and a transfer can be initiated by sending data over the Address lines. This signal will clear after three Address Ready signals and will not set again until the last word of the complete channel transfer has been stored in Central Memory. Transmit Address will clear if there is an error condition on the CP side and in that case Transmit Address will not set again until Clear Channel is sent.

MEMORY CHANNEL FUNCTIONS FOR INPUT FROM CPU

The following describes the input operation functions for the Memory Channel. The functions are listed below.

HIA : 0§	Clear channel busy, done flags
HIA : 1	Enter I/O Memory address
HIA : 2	Enter upper Central Memory address
HIA : 3	Enter lower Central Memory address
HIA : 4§	Read Central Memory, enter block length
HIA : 6§§	Clear interrupt enable flag
HIA : 7§§	Set interrupt enable flag
HIA : 14	Enter diagnostic mode

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10)

Interface Registers

Three interface registers are used to control the transfer. The Central Memory starting address is held in a 22-bit register. Two separate commands, HIA : 2 and HIA : 3 load the 22-bit address. The I/O Processor I/O Memory starting address is held in a 16-bit register that is loaded by an HIA : 1 command. The third register holds 14 bits of the transfer word count, or block length, entered by the HIA : 4 function.

HIA : 0 - Clear Channel Busy, Done Flags

This function clears the channel busy and done flags. It will cause an error condition if issued while this channel is active. HIA : 0 must be used to clear an error condition on the channel.

HIA : 1 - Enter I/O Memory Starting Address

This function enters the content of the I/O Processor accumulator into the I/O Memory address register. The 2^0, 2^1 bits of the address are forced to 0. This command does not alter the state of the channel busy or done flags. If this function is given while this channel is active, an error conditiion occurs.

HIA : 2 - Enter Upper Central Memory Address

This function enters the 2^0 - 2^{12} bits of the I/O Processor accumulator into the 2^9 - 2^{21} positions of the Central Memory starting address register. The states of the channel busy and done flags are unaltered. This command will cause an error condition if issued while this channel is active.

HIA : 3 - Enter Lower Central Memory Address

This function enters the 2^0 - 2^8 bits of the I/O Processor accumulator into the 2^0 - 2^8 bits of the Central Memory starting address register. The states of the channel busy and done flags are unaltered. This command will cause an error condition if issued when the input channel is active.

HIA : 4 - Read Central Memory, Enter Block Length

This command enters the lowest 2^0 - 2^{13} bits of the I/O Processor accumulator into the block length register. (This value is treated as a count of 64-bit words with a zero value indicating the maximum 16,384 words.) The busy flag is set and the done flag is cleared and the transfer is initiated. Upon completion, the done flag will set and the busy flag will clear. If during the transfer an unrecoverable error occurs, the transfer will terminate and the done flag will set and the busy flag will remain set. This command will cause an error condition if issued while the input channel is active.

HIA : 6 - Clear Interrupt Enable

This function clears the interrupt enable flag for the channel. The states of the busy and done flags remain unaltered.

HIA : 7 - Set Interrupt Enable

This function sets the interrupt enable flag for the channel. The states of the busy and done flags remain unaltered.

HIA : 14 - Enter Diagnostic Mode

This function enters the 2^0 - 2^2 bits of the I/O Processor accumulator into the diagnostic mode register. The modes are summarized in table 7-13. Only one mode is valid at a time. All diagnostic modes are cleared with a master clear, or alternately, by doing HIA : 0, followed by HIA : 14 with the accumulator = 0. Mode 0 is active until any other mode is selected. This function is for maintenance purposes only.

Table 7-13. Input channel diagnostic modes

Mode Designator	Function
0	Set Last Word flag (only if channel is active)
1	Disable Last Word flag from CPU
2	Force constant Address Ready
3	Disable third Address Ready
4	Transfer first address with a "0" parity bit, third address with a "1" parity bit, and disable Data Ready from CPU
5	Force Clear Channel without inactivating IOP
6	Disable Block Length = 0
7	Disable IOP data error correction and detection

Memory Channel Input Error Processing

If an irrecoverable error occurs in an input transfer from the Central Memory, the busy and done flags will both set. An error code is generated and sent to an error log. The error codes are summarized in table 7-14.

Memory Channel Input Sequence

Figure 7-10 shows the sequence of signals for transferring data from Central Memory to I/O Memory.

The only recoverable error is correctable data error. This type of error is ignored by the channel because the data is corrected automatically; the syndrome information is sent to an error log.

Table 7-14. Input channel error codes

Error Code	Name	Condition
1	Function Error	Indicates a function 0, 1, 2, 3 or 4 was issued while the channel was active.
2	Active Error	Indicates the CPU side went inactive while the IOP side was still active.
3	Transmit Address Timeout	Indicates an IOP input was initiated but the CPU did not send a transmit address within 2 milliseconds.
4	CPU Address Error	Indicates the CPU side received greater or less than three address readies or there was a parity error in one of the address channel transfers.
5	CPU IOP Data Error	Indicates the CPU side has a multiple data error from memory or the IOP side received data with a multiple error.
6	Block Length Error	Indicates the IOP received the last word and the block length count was not 0 or the last word was not received and the block length count was equal to 0.
7	Data Ready Timeout	Indicates the IOP did not receive data within 2 milliseconds.

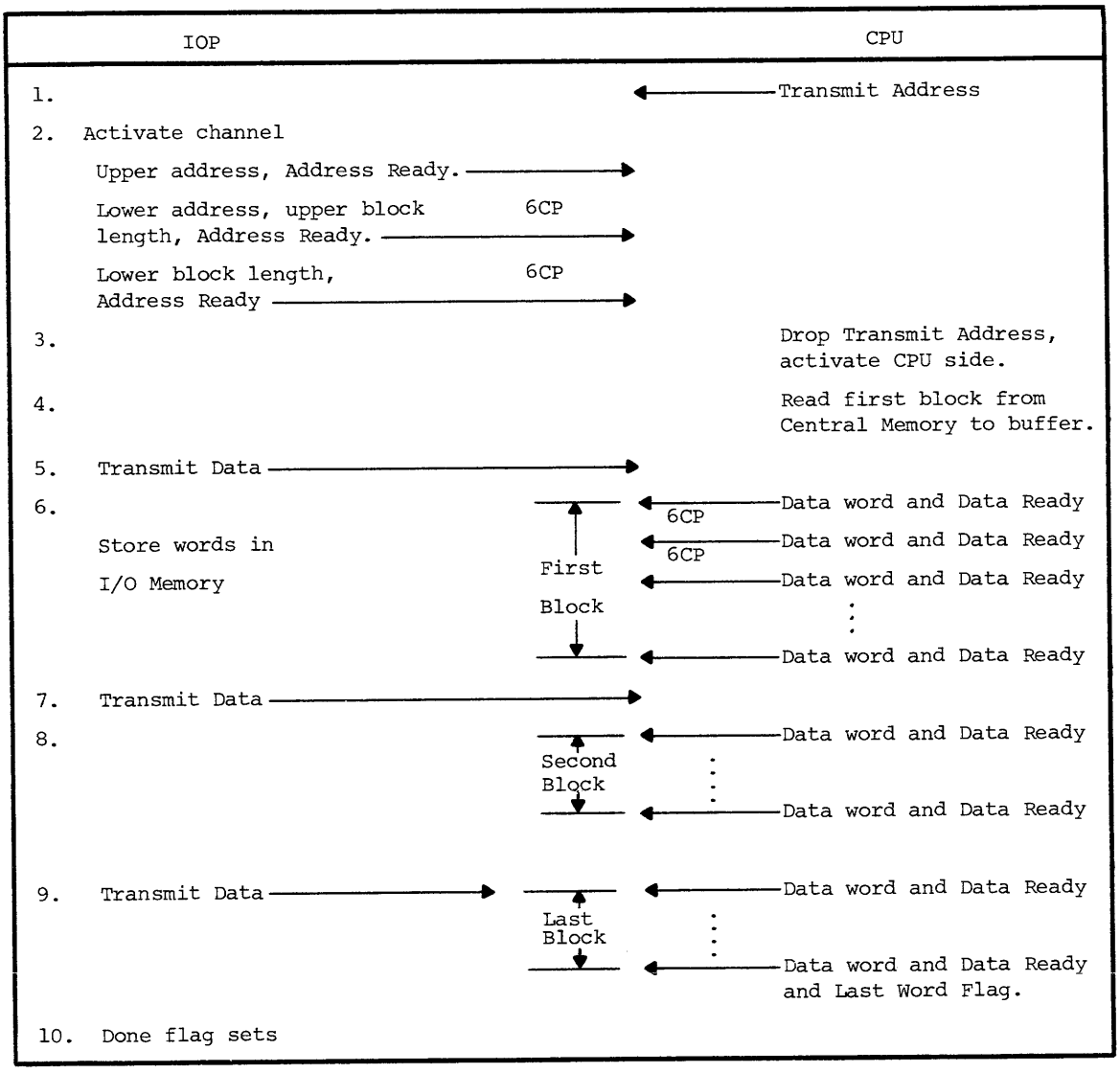


Figure 7-10. Memory Channel sequence, input to I/O Processor

MEMORY CHANNEL FUNCTIONS FOR OUTPUT TO CPU

The I/O Processor may have a channel for sending data to the CPU at over to 800 megabits per second. Transfers are done in blocks of 64-bit words under I/O Processor control. Error detection and correction are used to protect data quality. The functions are listed below.

HOA : 0§	Clear channel busy, done flags
HOA : 1	Enter I/O Memory address
HOA : 2	Enter upper Central Memory address
HOA : 3	Enter lower Central Memory address
HOA : 5§	Write Central Memory, enter block length
HOA : 6§§	Clear interrupt enable flag
HOA : 7§§	Set interrupt enable flag
HOA : 14	Enter diagnostic mode

Interface Registers

The interface has three registers used to control the output transfers. One 22-bit register holds the Central Memory starting address. The 22-bit address is loaded by the HOA : 2 and HOA : 3 functions. The I/O Processor I/O Memory starting address is held in a 16-bit register which is entered by the HOA : 1 function. A 14-bit register holds the transfer word count, or block length in 64-bit words, that is loaded by the HOA : 5 command.

HOA : 0 - Clear Channel Busy, Done Flags

This function idles the channel by clearing the busy and done flags. This function causes an error condition if issued while the output channel is active. This command must be used to clear an error condition in the channel.

HOA : 1 - Enter I/O Memory Address

This function enters the value of the I/O Processor accumulator into the I/O Memory address register. The 2^0 - 2^1 bits are forced to a zero value. The states of the busy and done flags are unaltered. This command causes an error condition if issued when the output channel is active.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

HOA : 2 - Enter Upper Central Memory Address

This function enters the 2^0 - 2^{12} bits of the I/O Processor accumulator into the 2^3 - 2^{15} bits of Central Memory starting address register. The states of the busy and done flag are unaltered. This function causes an error condition if issued when the output channel is active.

HOA : 3 - Enter Lower Central Memory Address

This function enters the 2^0 - 2^8 of the I/O Processor accumulator into the 2^0 - 2^8 bits of the Central Memory starting address register. The states of the busy and done flags are unaltered. This command causes an error condition if issued when the output channel is active.

HOA : 5 - Write Central Memory, Enter Block Length

This function enters the 2^0 - 2^{13} bits of the I/O Processor accumulator into the block length register. (This value is treated as a count of 64-bit words with a zero value indicating the maximum 16,384 words.) The busy flag is set and the done flag is cleared and the transfer is initiated. Upon completion, the done flag will set and the busy flag will clear. If during the transfer an irrecoverable error occurs, the transfer will terminate and the done flag will set and the busy flag will be left set. This command will cause an error condition if issued while the output channel is active.

HOA : 6 - Clear Interrupt Enable

This function clears the interrupt enable flag. The states of the busy and done flags are unaltered.

HOA : 7 - Set Interrupt Enable

This function sets the interrupt enable flag. The states of the busy and done flags are unaltered.

HOA : 14 - Enter Diagnostic Mode

This function enters the 2^0 - 2^7 bits of the I/O Processor accumulator into the diagnostic mode register. The modes are summarized in table 7-15. This is only for maintenance purposes.

With the exception of mode 6, only one mode is valid at a time. Mode 6 is a special case. It is held if it has been set and cannot be cleared by entering another mode. All diagnostic modes clear with master clear, or alternately by doing HOA : 0, followed by HOA : 14 with the accumulator = 0. Mode 0 is active until any other mode is selected.

Table 7-15. Output channel diagnostic modes

Mode Designator	Function
0	Set Last Word flag (only if channel is active)
1	Disable Last Word flag from CPU
2	Force constant Address Ready
3	Disable third Address Ready
4	Transfer first address with a "0" parity, third address with a "1" parity bit, and disable Data Ready to CPU
5	Force Clear Channel without inactivating IOP
6	First word 2^{56} - 2^{63} bits used as second word check byte, third word 2^{56} - 2^{64} bits as fourth word check byte, etc.
7	Disable error correction and detection as CPU

Central Memory Output Error Processing

If an irrecoverable error occurs in an output transfer to the Central Memory, the busy and done flags both set. An error code is generated and sent to an error log. The error codes are summarized in table 7-16.

The only recoverable error is a correctable data error. This type of error is ignored by the channel interface because it is corrected automatically. The syndrome information is sent to an error log.

Table 7-16. Output channel error codes

Error Code	Name	Condition
1	Function Error	Indicates a function 0, 1, 2, 3 or 5 was issued while the channel was active.
2	Active Error	Indicates the CPU side went inactive while the IOP side was still active.
3	Transmit Address Timeout	Indicates an IOP input was initiated but the CPU did not send a transmit address within 2 milliseconds.
4	CPU Address Error	Indicates the CPU side received greater or less than three address readies or there was a parity error in one of the address channel transfers.
5	CPU Data Length Error	Indicates the CPU received data with a multiple error or the last word was received and the block length count was not 0 or the last word was not received and the block length count was equal to 0.
6	Last Word Timeout	Indicates the last word was sent but the CPU did not go inactive within 2 milliseconds.
7	Transmit Data Timeout	Indicates a transmit data was not received within 2 milliseconds.

Memory Channel Output Sequence

Figure 7-11 shows the sequence of signals for transferring data from I/O Memory to Central Memory.

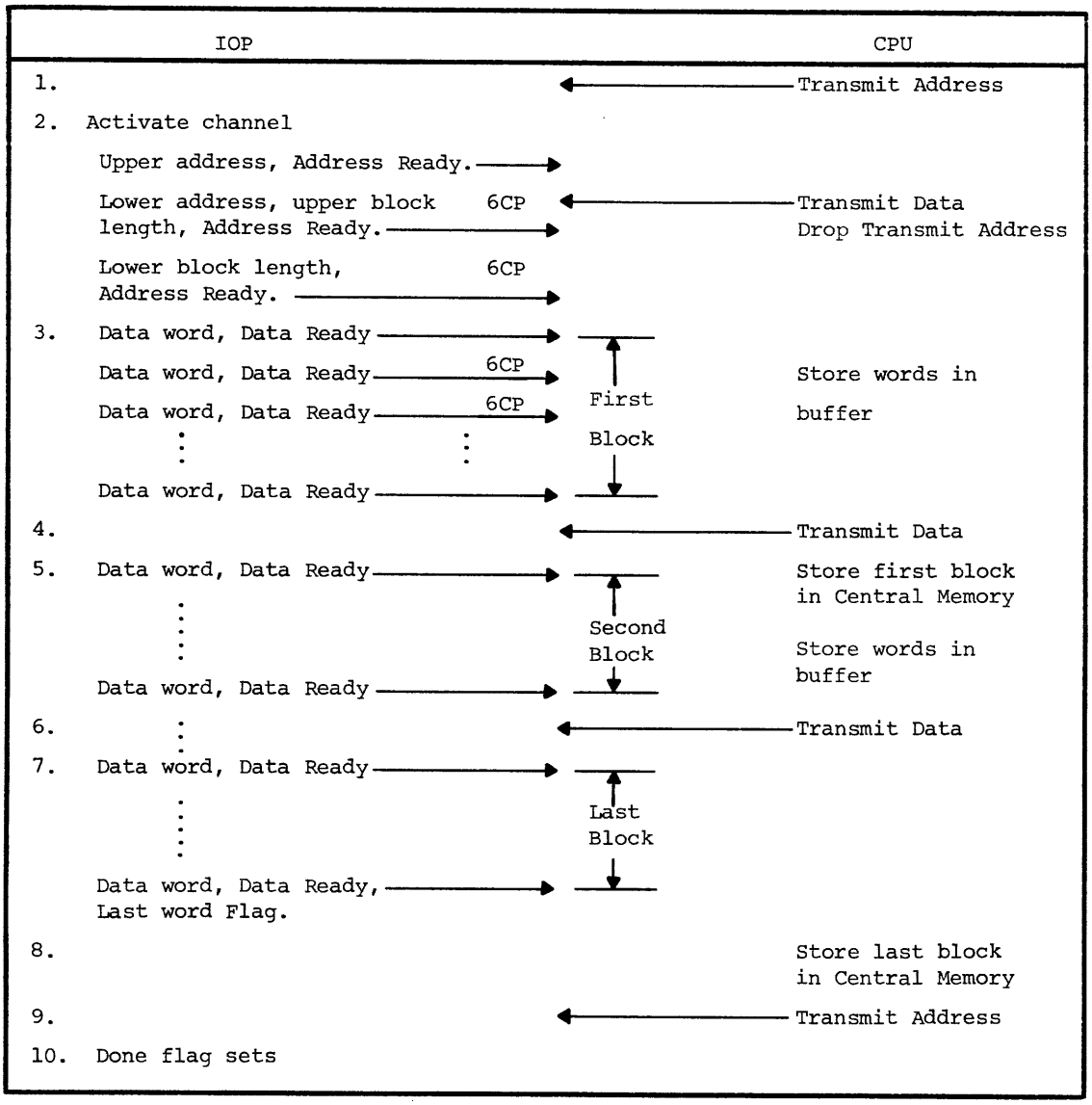


Figure 7-11. Memory Channel sequence, output from I/O Processor

ERROR LOGGING CHANNEL

The I/O Processor may have an error logging channel connected. This channel reports errors from the I/O Memories of three other I/O Processors, from the Buffer Memory, from the Central Memory, and from the Memory Channels to the Central Memory. If any error condition occurs in any of the reporting devices, the channel done flag sets. The functions are listed below.

ERA : 0§	Idle channel
ERA : 6§§	Clear interrupt enable flag
ERA : 7§§	Set interrupt enable flag
ERA : 10	Read error status
ERA : 11	Read error information (first parameter)
ERA : 12	Read error information (second parameter)
ERA : 13	Read error information (third parameter)

INTERFACE REGISTERS

The error log interface has four interface registers. The error status register is a 9-bit register that holds set bits for the device causing the error. This register is accessed by the ERA : 10 command.

The first parameter word register has 16 bits, and stores I/O Memory failing locations, or Buffer Memory error data, or Memory Channel error data.

The second parameter word register has 16 bits, and stores the lower address for the Buffer Memory or Central Memory failing locations.

The third parameter word register has 8 bits, and stores the upper address for either the Buffer Memory or the Central Memory.

ERA : 0 - IDLE CHANNEL

This function clears all error flags stored in the interface registers. The done flag is cleared.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

ERA : 6 - CLEAR INTERRUPT ENABLE FLAG

This function clears the interrupt enable flag for this channel.

ERA : 7 - SET INTERRUPT ENABLE FLAG

This function enables the interface to interrupt the I/O Processor.

ERA : 10 - READ ERROR STATUS

This function returns the contents of the interface error status register to the accumulator. The interpretation of the set bits is given in table 7-17.

Table 7-17. Error status register bits

Bit	Control Signal
2^0	I/O Processor 1 I/O Memory error
2^1	I/O Processor 2 I/O Memory error
2^2	I/O Processor 3 I/O Memory error
2^3	Buffer Memory error
2^4	Central Memory error
2^5	Memory Channel input A error
2^6	Memory Channel output B error
2^7	Memory Channel input C error
2^8	Memory Channel output D error

The selection of physical devices such as I/O Processors and Memory Channels as A, B, C, etc. is part of the overall system definition. Appendix F has a typical system definition.

Errors occurring in the I/O Processor having the error logging channel are reported to that I/O Processor via its own I/O Memory error channel (LME).

ERA : 11 - READ ERROR INFORMATION (FIRST PARAMETER)

This function returns an error parameter word to the accumulator, selected by a bit set in the accumulator when the function issues. Only one bit among the 2^0 - 2^8 positions may be set at one time. The accumulator bits select parameters as listed in table 7-18.

When an I/O Processor I/O Memory address is read, or when a Memory Channel error data word is read, the device bit is cleared in the status register.

The address invalid flag (2^{15}) in the Buffer Memory error data is set when the address that will be given in the second and third parameter words is not the correct address for the error being reported in the first parameter word. This can happen when a later read reference follows closely after a read reference that detects an error in its data. The address for the later read reference is incorrectly reported in the second and third parameter words, and should be disregarded.

This function must be immediately preceded by a logical product instruction as follows:

```
011777  A=A&d  d=777
171xxx  ERA : 11
        or
015xxx  A=A&k
xxx777  k=xxx777
171xxx  ERA : 11
```

ERA : 12 - READ ERROR INFORMATION (SECOND PARAMETER)

This function returns the content of the second parameter word register to the accumulator. The second parameter word is selected by bits 2^3 or 2^4 in the accumulator when the function is sent. Only one bit may be set in the accumulator 2^3 and 2^4 bit positions at a time. Selection is as follows.

2^3 Buffer Memory lower address (address bits 2^0 - 2^{15})

2^4 Central Memory lower address (address bits 2^0 - 2^{15})

Table 7-18. First error parameter selection

Error Status Register Bit	First Parameter Information Returned
2^0	I/O Processor A I/O Memory failing address: 2^0 - 2^1 Bank number 2^2 - 2^3 Section number 2^4 Byte: 0 Bits 2^0-2^7 1 Bits 2^8-2^{15}
2^1	I/O Processor B I/O Memory failing address, same format as for 2^0.
2^2	I/O Processor C I/O Memory failing address, same format as for 2^0.
2^3	Buffer Memory error data: 2^0 - 2^7 Syndrome bits 2^9 Correctable error if 1 2^{10} Noncorrectable error if 1 2^{12} - 2^{13} Port number 2^{15} Address invalid if 1
2^4	Central Memory error data: 2^0 - 2^7 Syndrome bits 2^9 Correctable error 2^{10} Noncorrectable error 2^{12} - 2^{13} Read mode: 0 Scalar 1 I/O 2 Vector 3 Fetch
2^5	Input Memory Channel A error data: 2^0 - 2^7 Syndrome bits 2^8 - 2^{10} Error code (refer to Central Memory input error processing, this section)
2^6	Output Memory Channel B error data, same as for 2^5. Refer to Central Memory output error processing, this section, for error codes.
2^7	Input Memory Channel C error data, same format as for 2^5.
2^8	Output Memory Channel D error data, same format as for 2^5. Refer to Central Memory output error processing, this section, for error codes.

This function must be immediately preceded by a logical product instruction as follows:

```
011777  A=A&d  d=777
172xxx  ERA : 12
        or
015xxx  A=A&k
xxx777  k=xxx777
172xxx  ERA : 12
```

ERA : 13 - READ ERROR INFORMATION (THIRD PARAMETER)

This function returns the content of the third parameter word register to the accumulator. The third parameter word is selected by bits 2^3 or 2^4 in the accumulator when the function issues. Only one bit may be set in the field at a time. Selection is as follows.

- 2^3 Buffer Memory upper address (address bits 2^{16} - 2^{22} going into accumulator positions 2^0 - 2^6)
- 2^4 Central Memory upper address (address bits 2^{16} - 2^{21} going into accumulator positions 2^0 - 2^5)

The reading of the third parameter word clears the error bits in the status register for the Central Memory or for the Buffer Memory.

This function must be immediately preceded by a logical product instruction as follows:

```
011777  A=A&d  d=777
173xxx  ERA : 13
        or
015xxx  A=A&k
xxx777  k=xxx777
173xxx  ERA : 13
```

BLOCK MULTIPLEXER CHANNEL

The I/O Subsystem communicates with IBM-compatible equipment over the block multiplexer channels. The following functions are used.

BMA : 0	Clear channel control
BMA : 1	Send reset function
BMA : 2	Channel command
BMA : 3	Read REQUEST-IN address
BMA : 4	Asynchronous I/O
BMA : 5	Delay counter diagnostic
BMA : 6§§	Clear channel interrupt enable flag
BMA : 7§§	Set channel interrupt enable flag
BMA : 10	Read I/O Memory address
BMA : 11	Read byte counter status
BMA : 12	Read status/address
BMA : 13	Read input tags
BMA : 14	Enter I/O Memory address
BMA : 15	Enter byte count
BMA : 16	Enter device address/mode
BMA : 17	Enter output tags

GENERAL CHARACTERISTICS

Block multiplexer channels are grouped into sets of four channels, which share one Cray Research BMC-4 Block Multiplexer Controller. The controller interfaces to the Auxiliary I/O Processor (XIOP) via one DMA port. The speed capacity of the DMA port is shared among the four channels of the controller. The four channels operate asynchronously and compete only for I/O Memory references.

---

§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

A block multiplexer channel operates in either selector channel mode, byte multiplexer mode or block multiplexer mode. All IBM commands are possible. The channel features command and data chaining, and detection of RETRY STATUS, as well as the I/O ERROR ALERT and the HIGH SPEED option.

The channel drives one or more IBM-compatible control units, which in turn drive peripheral devices.

This manual does not explain or document the IBM communications protocol, the signals used, nor their sequencing. Refer to the appropriate IBM publications for detailed information and definitions of IBM terminology.

TRANSFER RATES

The block multiplexer channel data rate is determined by cable length and signal turn-around time within the control unit. For each byte of data or control information that is sent, an appropriate response signal or signals must be received. This results in a data rate limit of about 6.4 million bits a second, or 800 kilobytes a second. IBM-compatible peripheral controllers that use the high speed option (requiring an additional pair of control lines) may achieve about 12.8 million bits a second, or 1.6 million bytes a second.

DATA HANDLING

One important feature of the channel is the assembly and disassembly of data between the 8-bit channel and the 16-bit memory parcels. Figure 7-12 shows the data changes.

The BMC-4 reads four 16-bit parcels from the I/O Memory and sends out eight 8-bit bytes to the block multiplexer channel. Four more 16-bit parcels are read into buffers in the controller while the channel is transmitting the first group.

The BMC-4 reads eight 8-bit bytes from the channel and writes four 16-bit parcels into I/O Memory. If a read operation from the channel terminates with a byte length that does not evenly comprise four parcels, the last I/O Memory write will include unpredictable data in the last parcels.

RECORD SIZE

Data records may be any non-zero integer number of bytes in length. However, the data chaining feature must be used for lengths greater than 65,535 bytes.

In cases such as IBM tape records, an odd length header field can be read and stored at one memory area and the following data record can be stored at a different memory area beginning at a Buffer Memory word boundary. The data chaining feature permits a single large record to be broken into any size convenient for storage in Buffer Memory. This is under program control by means of I/O functions to the block multiplexer channel.

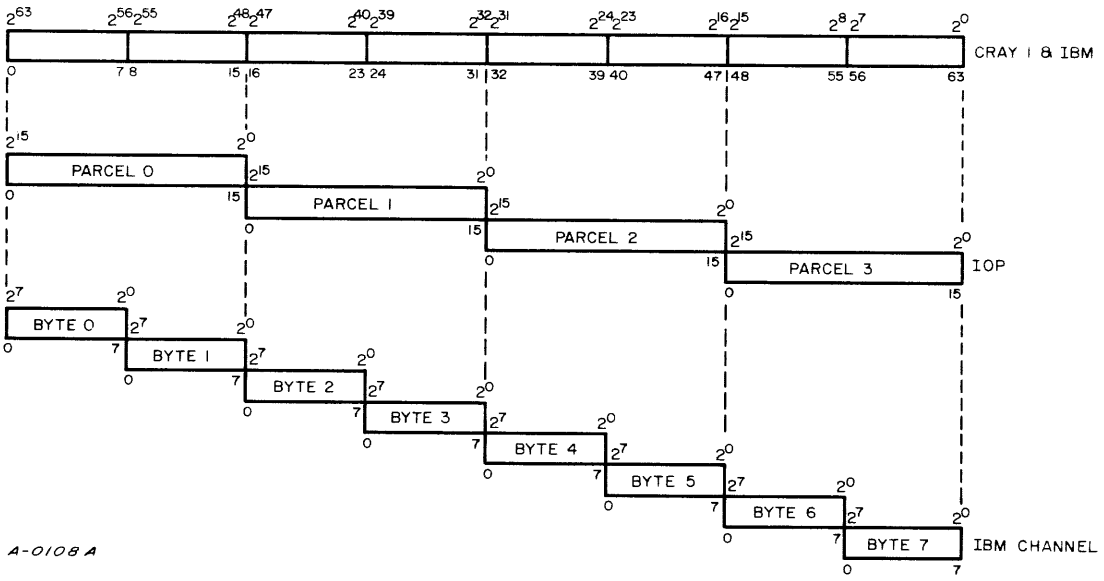


Figure 7-12. BMC-4 data assembly/disassembly

PARITY

Odd parity is checked on all address, status, and data inputs to the I/O Processor. Odd parity is generated for all address, control, or data outputs from the I/O Processor.

INTERRUPTS

All interrupts can be enabled or disabled for any block multiplexer channel. If interrupts are enabled, and a function in the range BMA : 1 through BMA : 5 completes, then an interrupt request is set. If an interrupt is selected for an input tag line such as REQUEST-IN, the interrupt request sets when the REQUEST-IN line goes to a logical 1.

When data chaining is enabled, an interrupt request is set each time the byte counter decrements to 0 and memory references are complete through that particular block of data.

BMA : 0 - CLEAR CHANNEL BUSY AND DONE FLAGS

This function clears the channel busy and done flags and all output tags except OP-OUT and SUP-OUT. No parameters are required for this action. It also clears interrupt conditions, provided interrupts are disabled by a BMA : 6 function. Since this function cannot be interlocked using the channel done flag, allow a 12-CP delay before issuing the next function to that particular channel.

BMA : 1 - SEND RESET FUNCTION

Several reset functions (table 7-19) perform various functions required by the equipment. Bits 2^1 and 2^0 of the accumulator content are used as a parameter that selects the specific function.

Table 7-19. Send reset function parameters

Parameter 2^1 - 2^0	Function
0 0	Clear all output tag lines
0 1	INTERFACE DISCONNECT
1 0	SELECTIVE RESET
1 1	SYSTEM RESET

Parameter xxxxx0 - Clear Output Tag Lines

This function clears all output tag lines and clears BUS 0 Out lines. The channel initially clears the channel done flag and sets the channel busy flag. Upon completion the channel done flag sets and the channel busy flag clears.

Parameter xxxxx1 - INTERFACE DISCONNECT

This function performs the INTERFACE DISCONNECT function to the currently selected control unit. The function clears the channel done flag and sets the channel busy flag. The control unit removes all signals from the Cray I/O Subsystem channel. When the control unit reaches the normal ending point in its sequence, it attempts to obtain selection in order to present any generated status to the Cray channel. The control unit does not generate any status as a result of the INTERFACE DISCONNECT.

The device path for the peripheral remains busy after it receives an INTERFACE DISCONNECT during an operation until the device-end status is accepted by the Cray block multiplexer channel.

The function should complete in about 7 microseconds. At that time the Cray channel done flag sets and the busy flag clears.

If the OPERATIONAL-IN signal from the equipment does not clear within 6 microseconds from the function issue, the Cray channel done flag sets and the busy flag remains set to indicate the error condition.

Parameter xxxxx2 - SELECTIVE RESET

This performs the SELECTIVE RESET function. The channel done flag clears and the channel busy flag sets. The function causes the OPERATIONAL-IN signal to clear and resets the currently selected peripheral device, along with its status. The current operation proceeds to a normal stopping point, and no data is transferred after the stop.

Only the peripheral device currently operating is affected. The device-end status is retained for transfer to the Cray channel after the reset.

The ready or not ready state of the control unit is generally not changed by the SELECTIVE RESET function.

The function should complete in about 7 microseconds, at which time the Cray channel busy flag clears and the done flag sets.

Parameter xxxxx3 - SYSTEM RESET

The SYSTEM RESET function clears the OPERATIONAL-IN signal and resets all control units along with their attached peripheral devices. The peripheral device statuses are also reset. The channel done flag clears and the channel busy flag sets at the beginning of execution.

The function should complete in about 6 microseconds at which time the Cray channel done flag sets and the busy flag clears. Input tag lines that are not reset by the SYSTEM RESET function cause the channel busy flag to remain set when the channel done flag sets.

BMA : 2 - CHANNEL COMMAND

This function sends commands to the control units. The specific command is selected by the accumulator bits at the time the function issues. Many of the commands require prior functions to set up interface registers, such as the I/O Memory address register, the byte count register, and the device address register. The command parameter bits are shown in table 7-20. The channel done flag clears and the channel busy flag sets at the beginning of execution. At completion of the function, the channel busy flag clears and the channel done flag sets.

Table 7-20. Channel command function parameter bits

Parameter Bit	Purpose
2^0	IBM-type control unit command bit 2^0
2^1	IBM-type control unit command bit 2^1
2^2	IBM-type control unit command bit 2^2
2^3	IBM-type control unit command bit 2^3
2^4	IBM-type control unit command bit 2^4
2^5	IBM-type control unit command bit 2^5
2^6	IBM-type control unit command bit 2^6
2^7	IBM-type control unit command bit 2^7
2^8	Unused
2^9	Unused
2^{10}	Unused
2^{11}	Unused
2^{12}	Unused
2^{13}	Unused
2^{14}	Unused
2^{15}	Unused

Parameter Command Bits

Bits 2^0 - 2^7 are the command bits for the BMA : 2 parameter. The command is for an IBM-type control unit. The parameter bits are shown in table 7-21.

Table 7-21. Channel command bit assignments

	P	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
TEST I/O	1	0	0	0	0	0	0	0	0
SENSE	P	M	M	M	M	0	1	0	0
READ BACKWARD	P	M	M	M	M	1	1	0	0
WRITE	P	M	M	M	M	M	M	0	1
READ	P	M	M	M	M	M	M	1	0
CONTROL	P	M	M	M	M	M	M	1	1

M = Modifier bit
P = Parity bit

The specific modifier codes and the particular mode set is dependent on the particular IBM-type control unit and the peripheral device used. The command byte is sent only during the initial selection sequence. All commands except TEST I/O may require a data transfer to satisfy the function.

All commands begin with an initial selection sequence which ends with either an ending status (channel-end or channel-end with device-end), a zero status, or an error status. To a command calling for transfer of status, control or data bytes, a zero status signifies that the transfer may begin.

A data-type transfer ends when the byte count decrements to 0 and no data chaining condition exists. The data-type transfer also ends when STATUS-IN is received in response to SERVICE-OUT or DATA-OUT.

The channel done flag is not set until processing of the control sequence is completed and data is transferred to or from I/O Memory.

Figure 7-13 shows a typical channel sequence for a read to the I/O Processor.

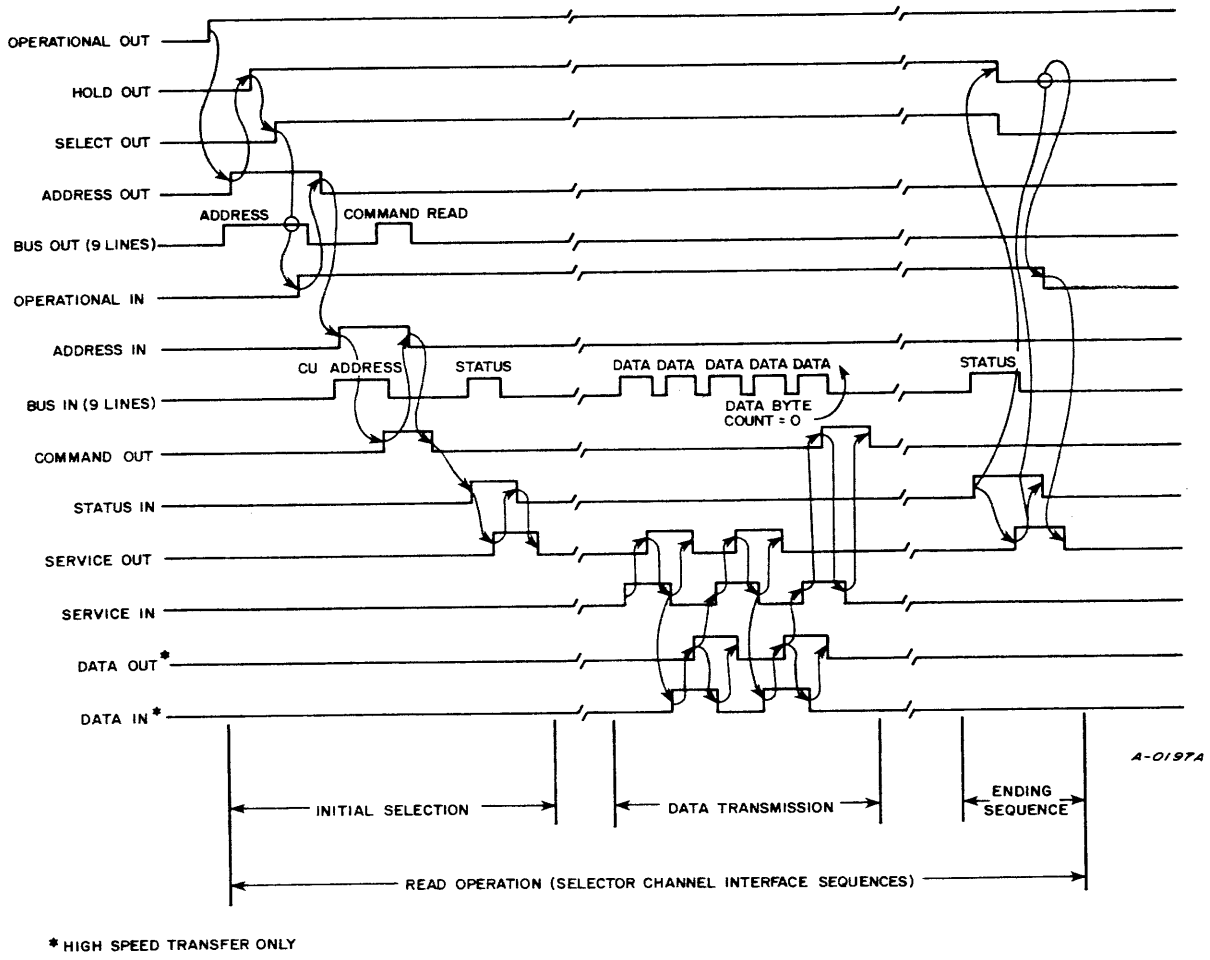


Figure 7-13. Channel read sequence

BMA : 3 - READ REQUEST-IN ADDRESS

This function first clears the channel done flag and sets the channel busy flag. If REQUEST-IN is detected, the channel accepts the requesting address. The address is checked for valid parity, the channel done flag is set, and the channel busy flag is cleared.

If REQUEST-IN is not detected, the channel done flag is set and the channel busy flag stays set. The address miscompare status is to be ignored if undetermined addresses are expected.

Figure 7-14 illustrates a typical REQUEST-IN channel sequence.

BMA : 4 - ASYNCHRONOUS I/O

Issuing the BMA : 4 function clears the channel done flag and sets the channel busy flag. If STATUS-IN is present, status is saved in the status register. If the STACK flag has been presented, COMMAND-OUT is returned in place of SERVICE-OUT to indicate STOP to the channel. When the sequence completes, the channel done flag sets and the channel busy flag clears. Refer to figure 7-14 for the typical REQUEST-IN channel sequence. See figure 7-15 for the asynchronous data and status processing. See table 7-26 for accumulator parameter bits.

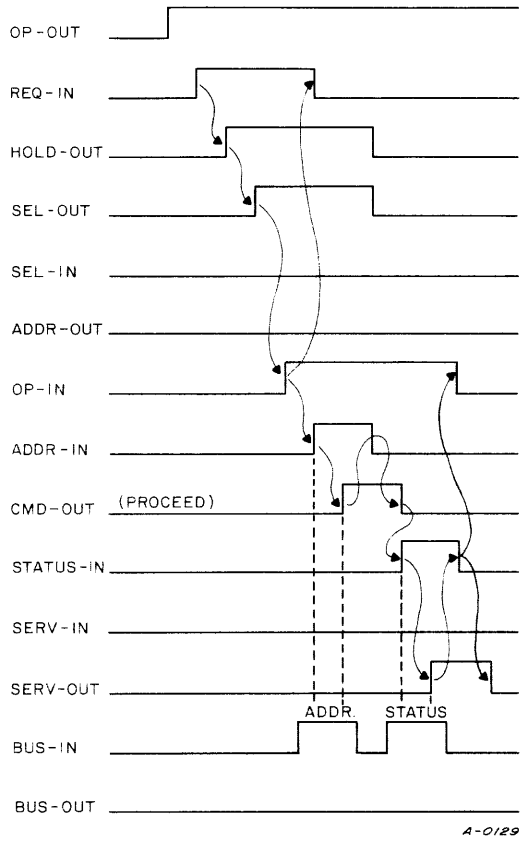


Figure 7-14. Channel ASYNCHRONOUS I/O sequence

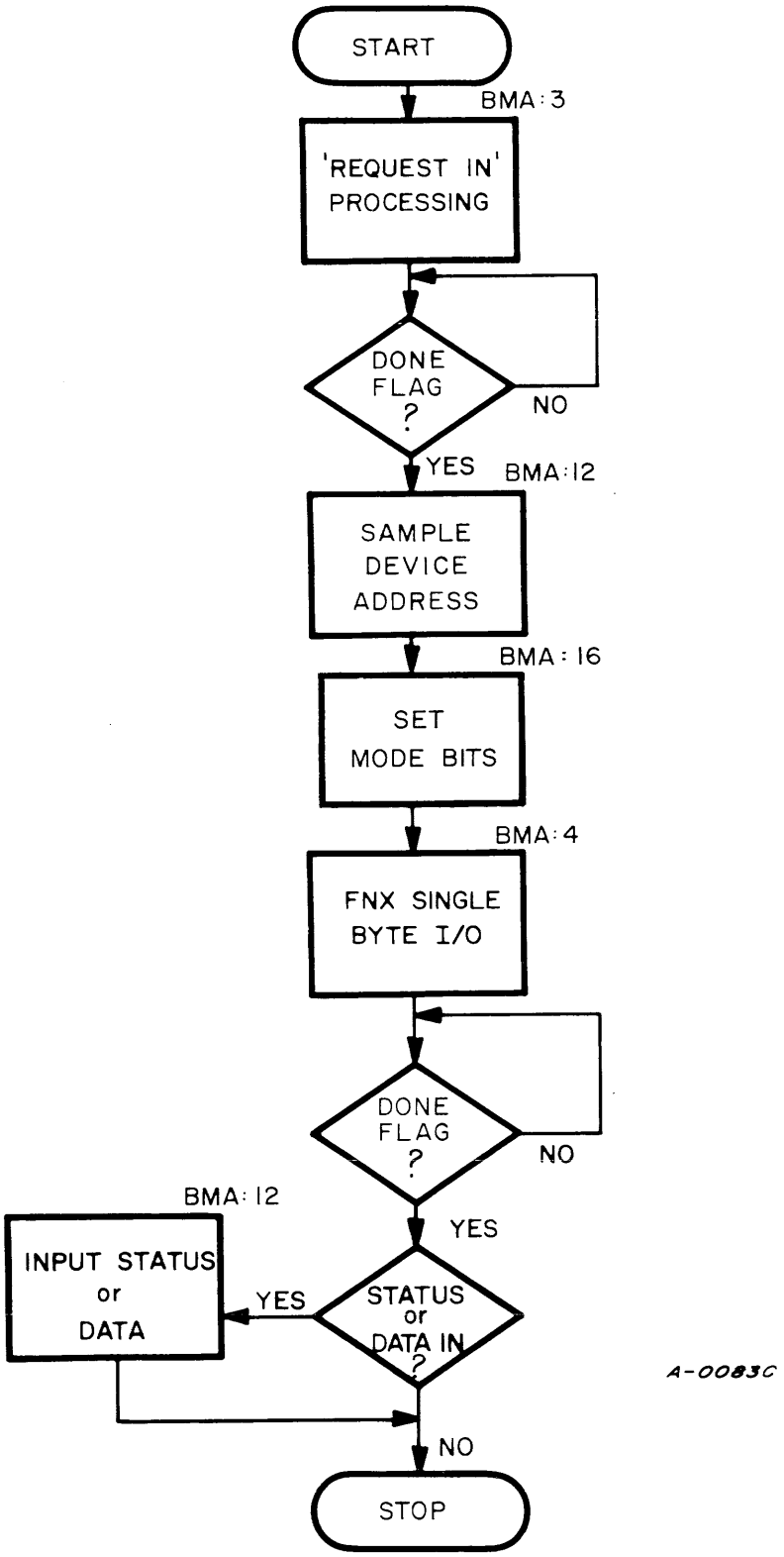


Figure 7-15. Asynchronous data and status processing

BMA : 5 - DELAY COUNTER DIAGNOSTIC

This function is for maintenance purposes only. It clears the channel done flag and sets the channel busy flag. Then the channel performs a 10 microsecond delay. After the delay times out, the channel done flag sets and the channel busy flag clears.

BMA : 6 - CLEAR CHANNEL INTERRUPT ENABLE FLAG

The BMA : 6 function clears the channel interrupt enable flag. This prevents the channel from interrupting the I/O Processor. The I/O Processor would in this case monitor the channel done flag to determine function completion. The channel busy and done flags are not affected by this function.

BMA : 7 - SET CHANNEL INTERRUPT ENABLE FLAG

This function sets the channel interrupt enable flag. This causes an I/O Processor interrupt for this channel whenever any of the following conditions occur:

- The channel done flag sets,
- The interrupt mode select, via a BMA : 16 function, has enabled an interrupt for an active input tag line, or
- During data chaining the byte counter has decremented to 0 and the last I/O Memory reference is complete for that segment of data.

BMA : 10 - READ I/O MEMORY ADDRESS

This function reads the current value in the I/O Memory address register and enters the value in the accumulator. The channel logic includes two I/O Memory address registers to support data chaining. The state of accumulator bit 2^0 when the function is issued determines which I/O Memory address register is to be read. Bit 2^0 of the value returned to the accumulator identifies from which register the I/O Memory address came.

Bit 2^1 in the value returned to the accumulator is set if data chaining is being used. Table 7-22 lists the accumulator content resulting from this function.

This function may be performed at any time relative to control functions.

During data chaining, interrupts occur after each buffer of data is transferred. This function clears the interrupt after a data transfer.

The BMA : 10 function has a programming restriction due to timing in the adder and shifter. The restriction applies if the instruction preceding a BMA : 10 function is any of the following: 4-7, 12, 13, 16, 17, 22, 23, 32, 33, 44-47, 52, 53, 62, or 63. In these cases an 011 or 015 logical product instruction, with the d or k fields set to all ones, should be inserted between the above instruction and the BMA : 10 function.

Table 7-22. Read I/O Memory address response bits

Accumulator Bit	Meaning
2^0	Register select status
2^1	Data chaining flag status (1=data chaining)
2^2	I/O Memory address 2^2
2^3	I/O Memory address 2^3
2^4	I/O Memory address 2^4
2^5	I/O Memory address 2^5
2^6	I/O Memory address 2^6
2^7	I/O Memory address 2^7
2^8	I/O Memory address 2^8
2^9	I/O Memory address 2^9
2^{10}	I/O Memory address 2^{10}
2^{11}	I/O Memory address 2^{11}
2^{12}	I/O Memory address 2^{12}
2^{13}	I/O Memory address 2^{13}
2^{14}	I/O Memory address 2^{14}
2^{15}	I/O Memory address 2^{15}

BMA : 11 - READ BYTE COUNTER

The byte counter records the number of bytes remaining in a data transfer. This counter is initially loaded with a value by a BMA : 15 load byte counter function. When the counter decrements byte-by-byte to 0, the channel interrupts the IOP and terminates the transfer. A transfer terminated by the control unit and not by decrementing to zero may result in a nonzero value in the counter. If data chaining is requested the byte counter is reloaded after decrementing to zero.

Due to the fixed timing in the IOP, this function must be executed twice in immediate succession to get a current value byte counter status to the IOP accumulator. The first execution moves the current byte counter status to the block multiplexer controller. The second execution moves the status to the IOP accumulator.

This function can be used to verify the accumulator fanout, the intermediate byte count status register in the block multiplexer controller, and the status path back to the IOP accumulator. Issuing a BMA : 15 function (explained later) loads the byte count into the byte counter status register in the block multiplexer controller. The next BMA : 11 function reads the byte count back from the status register to the IOP accumulator. The second BMA : 11 function performs as described above.

The channel busy and done flags are not affected by this function.

BMA : 12 - READ STATUS/ADDRESS

The status register holds the address and status mode bits read from the block multiplexer channel.

Due to the fixed timing in the IOP, this function must be executed twice in immediate succession to get valid current status/address information to the IOP accumulator. The first execution moves the current status/address to the block multiplexer controller. The second execution moves the status/address to the IOP accumulator.

This function can be used to verify the accumulator fanout, the intermediate status register in the block multiplexer controller, and the status path back to the IOP accumulator. Issuing a BMA : 16 function (explained later) loads address and mode bits into the block multiplexer controller status register. The next BMA : 12 function reads the status back to the IOP accumulator. The second BMA : 12 function performs as described above.

The channel busy and done flags are not affected by this function.

The status/address returned is the status/address information from the channel, and is entered in the IOP accumulator as shown in table 7-23.

Table 7-23. Status register bits

Accumulator Bits	Meaning
2^0	Status 2^0
2^1	Status 2^1
2^2	Status 2^2
2^3	Status 2^3
2^4	Status 2^4
2^5	Status 2^5
2^6	Status 2^6
2^7	Status 2^7
2^8	Address 2^0
2^9	Address 2^1
2^{10}	Address 2^2
2^{11}	Address 2^3
2^{12}	Address 2^4
2^{13}	Address 2^5
2^{14}	Address 2^6
2^{15}	Address 2^7

BMA : 13 - READ INPUT TAGS

This function reads the input tags from the channel to the I/O Processor accumulator.

Due to the fixed timing in the IOP, this function must be executed twice in immediate succession to get valid current input tags to the IOP accumulator. The first execution moves the current input tags to the block multiplexer controller. The second execution moves the input tags to the IOP accumulator.

This function can be used to verify the accumulator fanout, the intermediate output tags register, and the path back to the accumulator. Issuing a BMA : 17 function (explained later) loads the output tags into the output tags register. The next BMA : 13 function reads the stored output tags back to the IOP accumulator. The second BMA : 13 function brings the current input tags to the IOP accumulator.

The channel busy and done flags are not affected by this function.

The input tags status bits are shown in table 7-24.

Table 7-24. Input tags status bits

Accumulator Bits	Meaning
2^0	OPERATIONAL-IN
2^1	ADDRESS-IN
2^2	DISCONNECT-IN
2^3	SELECT-IN
2^4	REQUEST-IN
2^5	SERVICE-IN
2^6	DATA-IN
2^7	STATUS-IN
2^8	METERING-IN
2^9	MARK 0 IN
2^{10}	Unused - "0"
2^{11}	Data Buffer Pointer
2^{12}	Address Miscompare
2^{13}	Byte Count Zero
2^{14}	P-ROM Parity Error
2^{15}	BUS-IN Parity Error

BMA : 14 - ENTER I/O MEMORY ADDRESS

This function enters the current accumulator content into the I/O Memory address (IOMA) register. This address is the starting address in I/O Memory for the data transfer. The channel busy and done flags are not altered in the process. Two I/O Memory address registers are maintained for data chaining purposes, and are addressed by the 2^0 bit of the accumulator content. Data chaining must begin with register 0 and alternate between the 0 and 1 registers. Bit 2^1 of the accumulator content is the data chaining select flag for the chosen register and, if set, selects data chaining for that register/address.

Table 7-25 lists the accumulator bits for this function.

Table 7-25. I/O Memory address register bits

Accumulator Bits	Meaning
2^0	I/O Memory address register file select
2^1	Data chaining flag
2^2	I/O Memory address 2^2
2^3	I/O Memory address 2^3
2^4	I/O Memory address 2^4
2^5	I/O Memory address 2^5
2^6	I/O Memory address 2^6
2^7	I/O Memory address 2^7
2^8	I/O Memory address 2^8
2^9	I/O Memory address 2^9
2^{10}	I/O Memory address 2^{10}
2^{11}	I/O Memory address 2^{11}
2^{12}	I/O Memory address 2^{12}
2^{13}	I/O Memory address 2^{13}
2^{14}	I/O Memory address 2^{14}
2^{15}	I/O Memory address 2^{15}

BMA : 15 - ENTER BYTE COUNT

This function enters the accumulator content into the byte counter or into the next byte count register. The first BMA : 15 function following a BMA : 14 enters the accumulator content into the byte counter. Immediately following the first BMA : 15 function with a second BMA : 15 function enters the accumulator data into the next byte counter register. The transfer from the next byte count register to byte counter is done automatically between data segments in data chaining. Channel commands requiring no data, parameters, or status must establish a byte count of 0. The maximum count is 65,535 bytes. The channel busy and done flags are not altered in this process.

BMA : 16 - ENTER DEVICE ADDRESS/MODE

This function enters the accumulator content into the device address register. The device address register contains mode select bits and device address bits. The device address may be some combination of controller address bits and peripheral device address bits. The channel busy and done flags are not altered by this function. Table 7-26 shows the device address register bits.

Table 7-26. Device address register bits

Accumulator Bits	Meaning
2^0	Address/data out 2^0
2^1	Address/data out 2^1
2^2	Address/data out 2^2
2^3	Address/data out 2^3
2^4	Address/data out 2^4
2^5	Address/data out 2^5
2^6	Address/data out 2^6
2^7	Address/data out 2^7
2^8	Skip flag
2^9	Stack status flag
2^{10}	Command chaining mode select 2^0
2^{11}	Command chaining mode select 2^1
2^{12}	Interrupt mode select 2^0
2^{13}	Interrupt mode select 2^1
2^{14}	Channel mode select 2^0
2^{15}	Channel mode select 2^1

Parameter Mode Bits

Bits 2^8 - 2^{15} of the parameter are called mode bits. They are re-established during each BMA : 16 function. They operate as follows.

Skip flag - Bit 2^8 is the mode bit for the Skip flag. When set, it prohibits storing data into I/O Memory during read data transfers.

Stack status flag - Bit 2^9 is the mode bit for the stack status flag.

Command chaining mode select - Bits 2^{10} and 2^{11} select the mode in which command chaining will be used. Table 7-27 shows the translation of these mode bits.

Table 7-27. Command chaining mode selection

Parameter Bits 2^{11}-2^{10}	Selection
0 0	No chaining
0 1	Chain if channel-end status is detected
1 0	Chain if device-end status is detected
1 1	Chain if either channel-end status or device-end status is detected

Interrupt mode select - Parameter bits 2^{12} - 2^{13} select the mode in which interrupts are generated. Table 7-28 shows the translation of the interrupt mode bits.

Table 7-28. Interrupt mode selection

Parameter Bits 2^{13}-2^{12}	Selection
0 0	No interrupt mode selected, interrupts disabled
0 1	Interrupt on REQUEST-IN
1 0	Interrupt on STATUS-IN
1 1	Interrupt on DISCONNECT-IN

Channel type mode select - Parameter bits 2^{14} - 2^{15} are the mode bits that select which type of channel operation is to be used. The selection is shown in table 7-29.

Table 7-29. Channel type mode selection

Parameter Bits 2^{15}-2^{14}	Selection
0 0	Selector channel
0 1	Byte multiplexer channel
1 0	Block multiplexer channel
1 1	Reserved for future use

BMA : 17 - ENTER OUTPUT TAGS

This function enters the accumulator content into the output tags register, allowing direct program control of the output tags for special control sequences such as diagnostics. The channel busy and done flags are not altered by this function. Table 7-30 shows the accumulator bits for each output tag.

Table 7-30. Output tags register bits

Accumulator Bits	Meaning
2^0	OPERATIONAL-OUT
2^1	ADDRESS-OUT
2^2	HOLD-OUT
2^3	SELECT-OUT
2^4	COMMAND-OUT
2^5	SERVICE-OUT
2^6	DATA-OUT
2^7	SUPPRESS-OUT
2^8	METERING-OUT
2^9	MARK 0 OUT
2^{10}	Unused
2^{11}	Unused
2^{12}	CLOCK-OUT
$^{-13}$	Inhibit Parity Error
2^{14}	Force BUS-OUT Parity
2^{15}	Unused

PROGRAMMING EXAMPLES

The following examples illustrate two programming sequences for the block multiplexer channel. Example 1 shows the function sequence used to read a tape record of unknown length. Example 2 shows how to rewind the tape.

Example 1:

$Accumulator_8$	Function	Description
000000	BMA : 14	Set IOMA = 0 (arbitrary)
000000	BMA : 15	Set byte count = 0
000025	BMA : 16	Set device address (0-255)
000323	BMA : 2	Select GCR tape mode
	Wait for "Done"	
BUFA+2	BMA : 14	Set IOMA, chaining flag Buffer A
BUFB+3	BMA : 14	Set IOMA, chaining flag Buffer B
000005	BMA : 15	Set byte count Buffer A (header)
001000	BMA : 15	Set byte count Buffer B (1-65535)
000002	BMA : 2	Function read forward command
	Wait for interrupt	
BUFA+2	BMA : 14	Set IOMA, chaining flag, Buffer A
001000	BMA : 15	Set byte count Buffer A (1-65535)
	Wait for interrupt	Process header
BUFB+3	BMA : 14	Set IOMA, chaining flag Buffer B
001000	BMA : 15	Set byte count Buffer B
	Wait for interrupt	Transfer Buffer B to Buffer Memory
BUFA+2	BMA : 14	Set IOMA, chaining flag Buffer A
001000	BMA : 15	Set byte count Buffer A (1-65535)
	Wait for interrupt	Transfer Buffer A to Buffer Memory
	"Done" set	
	BMA : 11	
1000-X	BMA : 11	Read byte counter for residue transfer X bytes, Buffer B to Buffer Memory

Example 1 (continued):

$Accumulator_8$	Function	Description
	BMA : 12	
	BMA : 12	Verify device status
	BMA : 13	
	BMA : 13	Check input P.E. etc.

Example 2:

$Accumulator_8$	Function	Description
000000	BMA : 14	Set IOMA = 0 (arbitrary)
000000	BMA : 15	Set byte count = 0
000025	BMA : 16	Set device address (0-255)
000007	BMA : 2	REWIND function
	Wait for "Done"	
	Check for "Busy"	"Busy" set indicates error
	BMA : 12	
	BMA : 12	Verify device address and status
	BMA : 13	
	BMA : 13	Check for other errors

BUFFER MEMORY 8

INTRODUCTION

The Buffer Memory assists data transfer between peripheral devices and Central Memory. It is implemented as NMOS (negative channel metal oxide semiconductor) LSI storage circuits. Buffer Memory capacity ranges from 1 million to 8 million words of 64 bits each. The Buffer Memory is housed in the same chassis as the I/O Processors, in order to keep the intercabling delays small. Modules are of the same construction as the I/O Processor modules -- four printed circuit boards to each module. This section describes the speeds, organization, access, and addressing of the Buffer Memory.

MEMORY SPEEDS

The memory is a dynamic random access memory that refreshes its data once every 2 milliseconds. Refreshing is transparent and does not affect the random access capability. Access time is 200 nanoseconds. Memory cycle time is defined as the waiting time required after referencing a storage location before that location may be referred to again. The Buffer Memory cycle time is 40 CPs.

MEMORY ORGANIZATION

Two options are available for memory organization. The Buffer Memories with 1 to 4 million words use an 8-bank organization, and the 8-million word size uses the 16-bank phasing. The advantages of the 16-bank organization are that the number of bank conflicts (more than one reference to the same bank at the same time, causing one reference to wait) are greatly reduced and the memory bandwidth is doubled.

Each address in Buffer Memory contains 64 data bits and 8 bits of error correction information. Data is transferred to and from Buffer Memory in 16-bit parcels. Four parcels are stored in one 64-bit word. Parcels are received and sent in a 0,1,2,3 sequence. The packing arrangement is shown in figure 8-1.

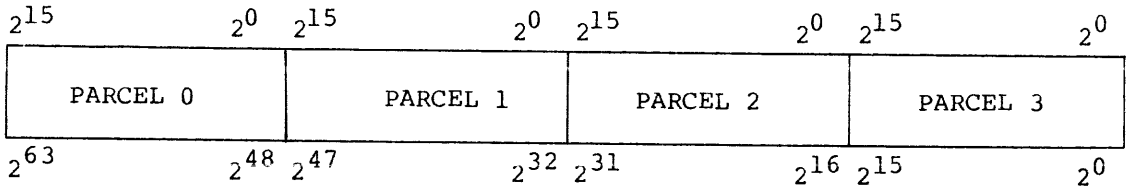


Figure 8-1. Parcel packing in memory word

MEMORY ACCESS

Buffer Memory has four ports, each of which can be connected to an I/O Processor, as shown in figure 8-2. An interface adapts one I/O Processor I/O Memory DMA port to one Buffer Memory port. Communication speed depends upon the number of banks in the Buffer Memory, the activity competing for the I/O Memory, and the number of other I/O Processors attempting to use the Buffer Memory. Each of the four Buffer Memory ports has access to all banks through a time-sharing scanner. If one port requests a reference to a bank that is busy, a reservation is made for the new port to gain access to the bank as soon as the bank becomes available. A similar scanning arrangement is used at the I/O Memory to share the six DMA ports.

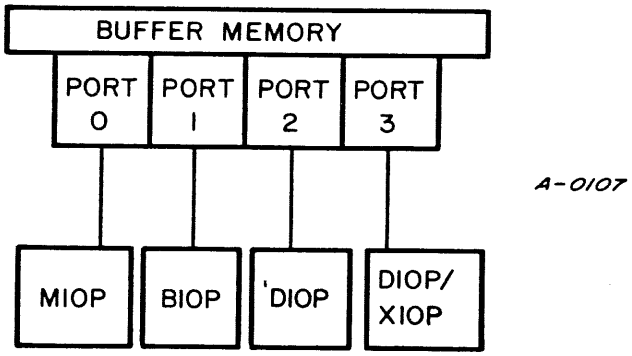


Figure 8-2. Buffer Memory port assignments

MEMORY ADDRESSING

The Buffer Memory capacity is either 1,048,576 or 4,194,304 or 8,388,608 words, requiring an address width of 23 bits. The 23-bit address is provided from an I/O Processor accumulator in two functions to the interface as shown in figure 8-3. The Buffer Memory does not address to the parcel level. The address represents a physical location in Buffer Memory.

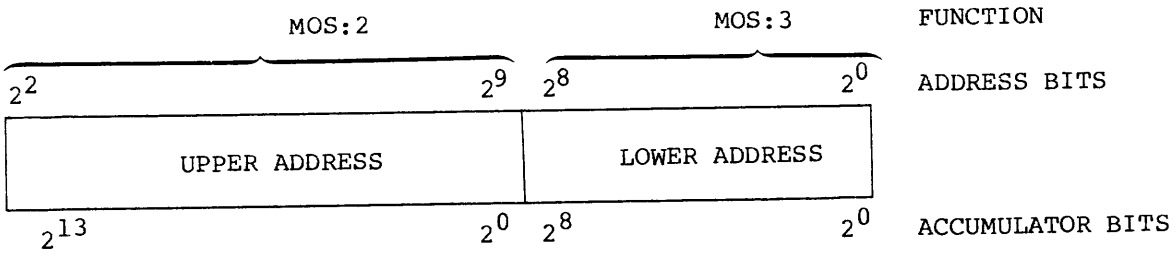


Figure 8-3. Buffer Memory address formation

ERROR PROTECTION

The Buffer Memory uses single error correction/double error detection (SECDED) logic to determine if the data has been altered by the storage cycle. When the data is written into memory, a checkword is generated for the word and stored with that word. The checkword is an 8-bit Hamming code. When the word is read from memory, the checkword and data word are processed to determine if any bits were altered. If no errors occurred, the word is passed to the I/O Processor.

If an error did occur, the 8 bits of the checkword are analyzed by the logic to find out if only 1 bit has been altered, or if more than 1 bit has changed. If it is only a single-bit error, the correction logic resets the bit in error to the correct state and passes the corrected word out to the I/O Processor.

If more than 1 bit of the stored word has been altered, the logic cannot correct the word. The interface signals the error condition by leaving the channel busy signal set after the block transfer. This interrupts the I/O Processor for an error handling routine. The error handling routine can include a call to the Master I/O Processor to have that processor retrieve the error information. This is explained in more detail in Channel for Error Logging in the Interfaces section.

The Buffer Memory uses the same SECDED logic design as used in the Central Memory. Refer to the Central Memory section for more details on error protection.

PART 4

APPENDIX SECTION

SUMMARY OF CPU TIMING INFORMATION A

When issue conditions are satisfied, an instruction completes in a fixed amount of time (memory references are exceptions). 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. One input path exists for each group of the four register groups (A, B, S, and T). The result register group input path must be free at the time the results would be stored. A previous instruction with a longer execution time could still be occupying the input path.

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.

Scalar instruction execution times in clock periods are given below.

where:
- A = A register
- B = B register
- C = Channel
- f = Floating-point
- I = Immediate
- lzc = Leading zero count
- M = Memory
- pop = Population count or population count parity
- RTC = Real-time clock
- ra = Reciprocal approximation
- S = S registers
- V = V registers
- VM = Vector mask

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

The following is an example of the use of this chart of execution times to optimize timing.

§ Issue may be delayed because of a functional unit reservation by a vector instruction. Memory may be considered a functional unit for timing considerations.

CAL Code			Execution Time	Reservations				
1	S1	S2+S3	3	S1				
2	A2	0 (immed.)	1	S1	A2			
3	S5	A2	2	S1		S5		
4	S4	S1+S3	3			S5	S4	
5	S6	S5&S1	1				S4	S6
6							S4	
7								

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 CPs. Memory is reserved for (VL)+5 CPs on a write operation, (VL)+4 CPs on a read operation.

2. The result register is reserved for the functional unit time +(VL) +2 CPs. The result register is reserved for the functional unit time +7 CPs if the vector length is less than 5. At functional unit time +2 (chain slot time) a subsequent vector instruction, which has met all other issue conditions, may issue. This process is called chaining. Several vector 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) CPs. Vector operand registers are reserved for 5 CPs 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 CPs.

If (Ak) is an even multiple of 8[§], results are produced every 4 CPs.

Functional unit	Time (CP)
Vector logical	2
Vector shift	4
Vector integer add	3
Floating-point add	6
Floating-point multiply	7
Reciprocal approximation	14
Memory	7
Vector population count	6

A transmit vector mask to Si (073) instruction is delayed by (VL) +6 CPs from the issue of a previous vector mask (175) instruction and is delayed by 6 CPs from the issue of preceding transmit (Sj) to VM (003) instruction.

HOLD ISSUE

A delay of issue results if a 100 - 137 instruction is the NIP register and a hold memory condition exists. The delay depends on the hold memory delay.

A delay of issue results if a 100 - 137 instruction is 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.

[§] Multiple of 4 for 8-bank phasing; refer to part 2 section 2.

Memory must be quiet before issue of the B and T register block copy instructions (034-037). The low-order 7 bits (Ai) affect the timing. Subsequent instructions may not issue for 14+(Ai) CPs if (Ai)≠0 and 5 CPs if (Ai)=0 when reading data to the B and T registers (034,036). They may not issue for 6+(Ai) CPs 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.

Conditional branch instructions cannot issue until an A0 or S0 operand register has been free for 2 CPs. Fall-through in buffer requires 2 CPs. Branch-in-buffer requires 5 CPs. When an "out of buffer" condition occurs the execution time for a branch instruction is 14 CPs§.

A 2-parcel instruction takes 2 CPs to issue.

Instruction issue is delayed 2 CPs when the next instruction parcel is in a different instruction parcel buffer. Instruction issue is delayed 12 CPs§§ if the next instruction parcel is not in an instruction parcel buffer.

HOLD MEMORY

A delay of 1, 2, or 3 CPs will be added to an A, B, S or T register 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 CPs. The scalar instruction address enters rank C at issue + 3 CPs.

§ 18 CPs for 8-bank phasing
§§ 16 CPs for 8-bank phasing

Scalar memory instruction timing (no conflict):

CP n	Issue, reserve Ai or Si register
CP n+1	Address rank A, sense conflict
CP n+2	Address rank B
CP n+3	Address rank C
.	
.	
.	
CP n+10	Clear register reservation
CP n+11	Complete

INTERRUPT TIMING

After a sensed interrupt condition, a minimum of 3 CPs + 2 parcel issues must occur before the interrupt is generated. During the first 3 CPs, if no hold issue conditions exist, instruction parcels may issue. At the end of the 3 CPs, the NIP register parcel is examined. If the NIP instruction is a 2-parcel instruction, 3 parcel issues occur before the interrupt. If the NIP instruction is a 1-parcel instruction, only 2 parcel issues occur before the interrupt.

PHYSICAL ORGANIZATION OF CPU B

MAINFRAME

The CPU mainframe is one of two sizes as shown in figure B-1. The logic chassis are arranged two in each column in an arc that is about 2.5 feet in radius. The larger mainframe with 12 columns extends 270° around the arc, while the 8-column version subtends 180° of the arc. The columns are about 6 1/2 feet tall. At the base of the columns, 1 1/2 feet high and extending outward about 2 1/2 feet, are cabinets for power supplies and cooling distribution systems.

Viewing the mainframe from the top, the upper chassis are labeled A through L (A through H in the 8-column version) proceeding counterclockwise. In the same manner, the lower chassis are named M through X (M through T in the 8-column version). The general chassis layout is shown in figure B-2. In the 8-column version, the I,J,K,L,U,V,W and X logic chassis are omitted.

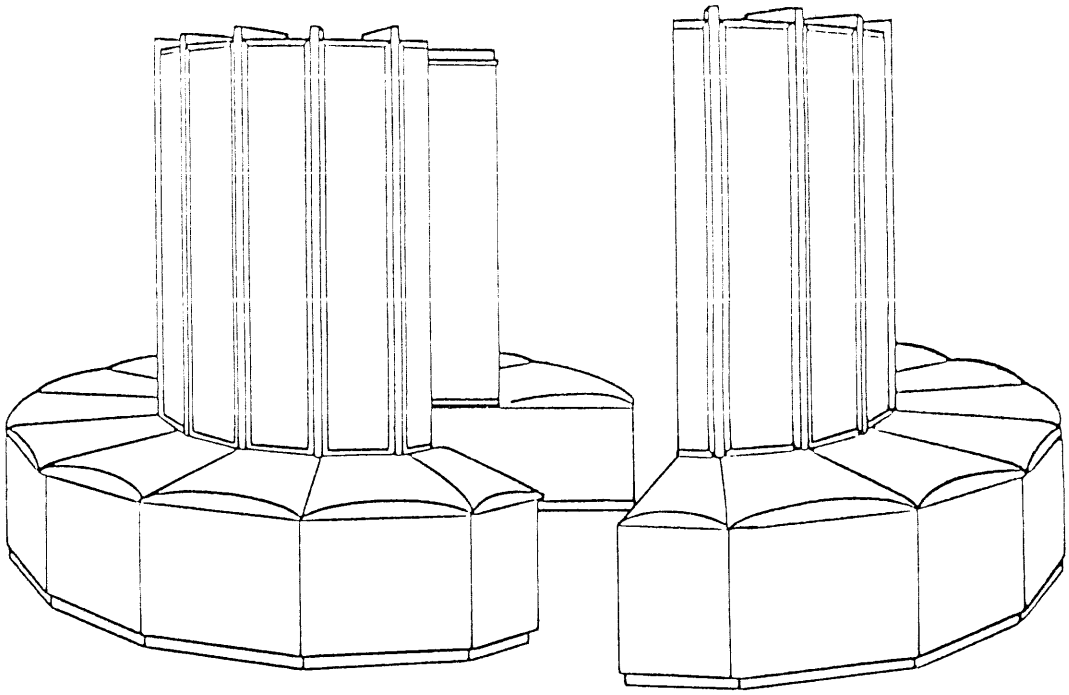
MODULES

The CRAY-1 computer system uses a basic module construction throughout 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.

A 2- to 4-million word CPU has 1684 modules. Modules are arranged 72 per chassis as illustrated in figure B-2. There are over 131 module types. Usage varies from 1 to 568 modules per type. 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, 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 memory banks.

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



- 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, or 16 chassis in 8 columns

- Approximately 1700 modules (12 column), 1100 for 8 column

- Approximately 130 standard module types

- 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

- Five memory options

- Weight 10,500 lbs (maximum memory size)

Figure B-1. Physical organization of CPU

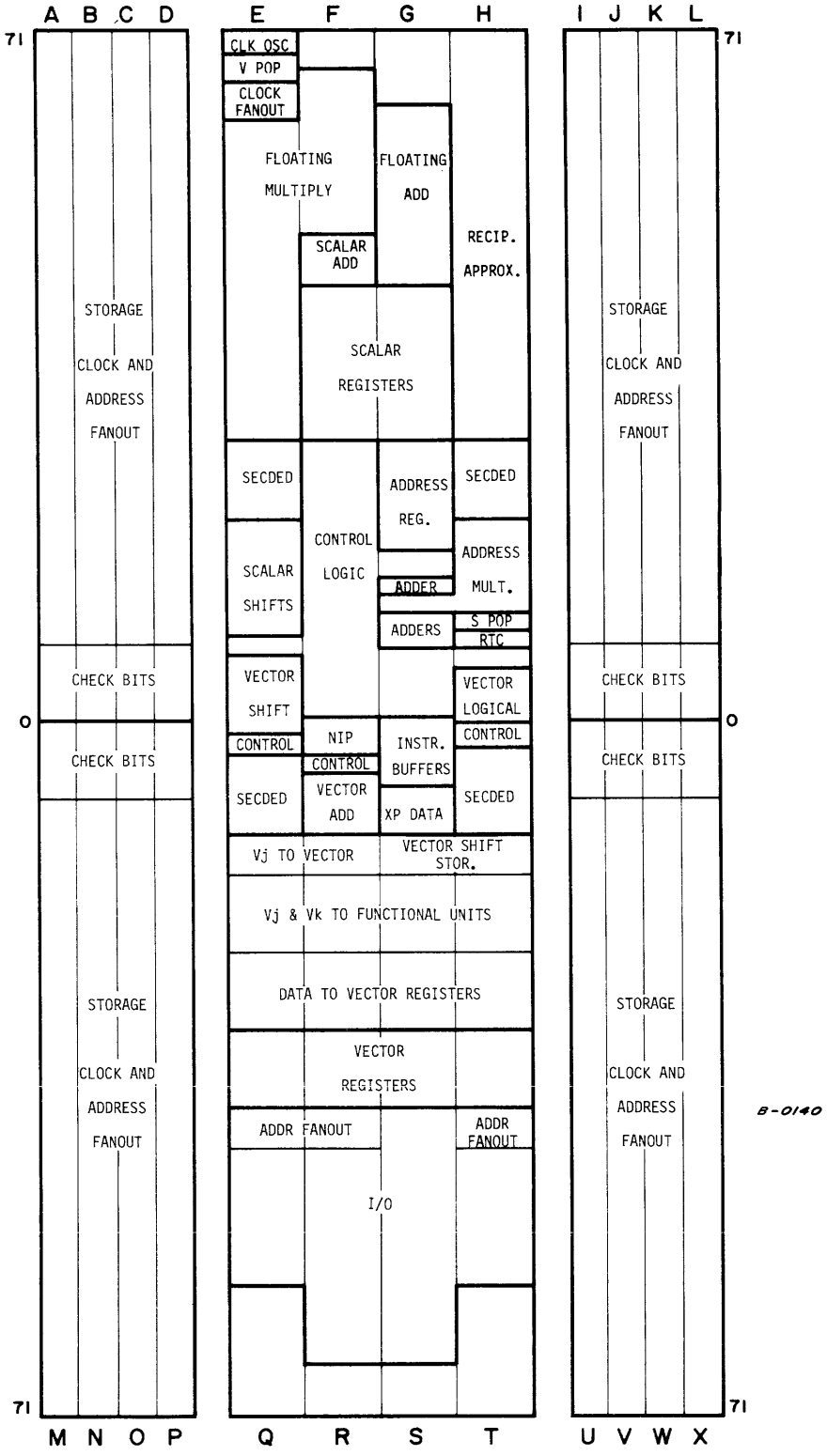


Figure B-2. General chassis layout

Each module has 96 pin pairs 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 used for trouble shooting. Test points are driven by circuits that do not drive other loads.

CLOCK

All timing within the CPU is controlled by a single-phase synchronous clock network. This clock has a period of 12.5 nanoseconds. All of the lines that carry the clock signal from the central clock source to the individual modules of the CPU are 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 3-nanosecond pulse (figure B-3) is formed on each module.

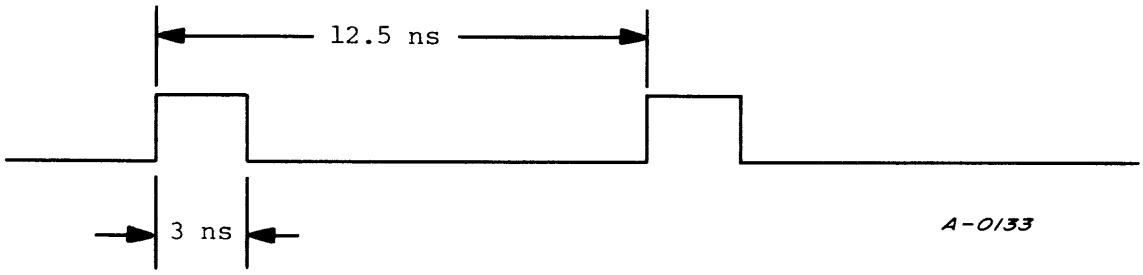


Figure B-3. Clock pulse waveform

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 CP 0. Thus, the third clock period would be referred to as CP 2.

POWER SUPPLIES

Thirty-six power supplies are used for the 12-column mainframe, or 24 for the 8-column models. There are twenty -5.2 volt supplies and sixteen -2.0 volt supplies (in the 12 column version). There are 12 of each voltage in the 8-column version. The supplies are divided into 12 groups (or 8) of 3, each group suppling one column. A logic column uses one -5.2 volt supply and two -2.0 volt supplies. A memory column uses two -5.2 volt supplies and one -2.0 volt supply. 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 12-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.

COOLING

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

SOFTWARE CONSIDERATIONS C

References to software in this publication are limited to those features of the CPU that provide for software or take it into consideration.

SYSTEM MONITOR

A monitor program is loaded at system deadstart and remains in Central Memory for as long as the system is used. Only the monitor program executes in CPU monitor mode and can execute monitor instructions. A program executing in monitor mode cannot be interrupted. A monitor program is designed to reference all of memory.

USER PROGRAM

A user program or 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. A user 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 Central Memory and possibly onto mass storage during system deadstart. Features of the operating system and organization of storage, which is a function of the operating system, is described in CRAY-OS Version 1 Reference Manual, CRI publication SR-0011.

SYSTEM OPERATION

System operation begins at CPU deadstart. Deadstart 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 deadstart sequence is initiated from the I/O Subsystem or the Maintenance Control Unit (MCU) depending on the model of the CRAY-1. The sequence is described in detail in part 2 section 3. During the deadstart sequence, a program containing an exchange package is loaded at absolute address 0 in the Central Memory. A signal from the MCU or I/O Subsystem 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 the vector merge instruction usage in subroutines such as TANGENT, 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.

CPU INSTRUCTION SUMMARY

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
000xxx	ERR		6-7	-	Error exit
†000ijk	ERR	exp	6-7	-	Error exit
††0010jk	CA,Aj	Ak	6-8	-	Set the channel (Aj) current address to (Ak) and begin the I/O sequence
††0011jk	CL,Aj	Ak	6-8	-	Set the channel (Aj) limit address to (Ak)
††0012jx	CI,Aj		6-8	-	Clear channel (Aj) interrupt flag
††0013jx	XA	Aj	6-8	-	Enter XA register with (Aj)
††0014j0	RT	Sj	6-10	-	Enter RTC register with (Sj)
††§0014j4	PCI	Sj	6-10	-	Enter interval register with (Sj)
††§0014j5	CCI		6-10	-	Clear PCI request
††§0014j6	ECI		6-10	-	Enable PCI request
††§0014j7	DCI		6-10	-	Disable PCI request
0020xk	VL	Ak	6-12	-	Transmit (Ak) to VL register
†0020x0	VL	1	6-12	-	Transmit 1 to VL register
0021xx	EFI		6-13	-	Enable interrupt on floating point error
0022xx	DFI		6-13	-	Disable interrupt on floating point error
003xjx	VM	Sj	6-14	-	Transmit (Sj) to VM register
†003x0x	VM	0	6-14	-	Clear VM register
004xxx	EX		6-15	-	Normal exit
†004ijk	EX	exp	6-15	-	Normal exit
005xjk	J	Bjk	6-16	-	Jump to (Bjk)
006ijkm	J	exp	6-17	-	Jump to exp
007ijkm	R	exp	6-18	-	Return jump to exp; set B00 to P
010ijkm	JAZ	exp	6-19	-	Branch to exp if (A0) = 0
011ijkm	JAN	exp	6-19	-	Branch to exp if (A0) ≠ 0
012ijkm	JAP	exp	6-19	-	Branch to exp if (A0) positive
013ijkm	JAM	exp	6-19	-	Branch to exp if (A0) negative
014ijkm	JSZ	exp	6-21	-	Branch to exp if (S0) = 0
015ijkm	JSN	exp	6-21	-	Branch to exp if (S0) ≠ 0
016ijkm	JSP	exp	6-21	-	Branch to exp if (S0) positive
017ijkm	JSM	exp	6-21	-	Branch to exp if (S0) negative
020ijkm	Ai	exp	6-23	-	Transmit exp = jkm to Ai
021ijkm	Ai	exp	6-23	-	Transmit exp = 1's complement of jkm to Ai
022ijk	Ai	exp	6-24	-	Transmit exp = jk to Ai
023ijx	Ai	Sj	6-25	-	Transmit (Sj) to Ai
024ijk	Ai	Bjk	6-26	-	Transmit (Bjk) to Ai
025ijk	Bjk	Ai	6-26	-	Transmit (Ai) to Bjk
026ij0	Ai	PSj	6-27	Pop/LZ	Population count of (Sj) to Ai
026ij1	Ai	QSj	6-27	Pop/LZ	Population count parity of (Sj) to Ai
027ijx	Ai	ZSj	6-28	Pop/LZ	Leading zero count of (Sj) to Ai
030ijk	Ai	Aj+Ak	6-29	A Int Add	Integer sum of (Aj) and (Ak) to Ai
†030i0k	Ai	Ak	6-29	A Int Add	Transmit (Ak) to Ai
†030ij0	Ai	Aj+1	6-29	A Int Add	Integer sum of (Aj) and 1 to Ai
031ijk	Ai	Aj-Ak	6-29	A Int Add	Integer difference of (Aj) less (Ak) to Ai

† Special syntax form
†† Priviledged to monitor mode
§ Programmable Clock Option only

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
†031i00	Ai	-1	6-29	A Int Add	Transmit -1 to Ai
†031i0k	Ai	-Ak	6-29	A Int Add	Transmit the negative of (Ak) to Ai
†031ij0	Ai	Aj-1	6-29	A Int Add	Integer difference of (Aj) less 1 to Ai
032ijk	Ai	Aj*Ak	6-31	A Int Mult	Integer product of (Aj) and (Ak) to Ai
033i0x	Ai	CI	6-32	-	Channel number to Ai (j=0)
033ij0	Ai	CA,Aj	6-32	-	Address of channel (Aj) to Ai (j≠0; k=0)
033ij1	Ai	CE,Aj	6-32	-	Error flag of channel (Aj) to Ai (j≠0; k=1)
034ijk	Bjk,Ai	,A0	6-34	Memory	Read (Ai) words to B register jk from (A0)
†034ijk	Bjk,Ai	0,A0	6-34	Memory	Read (Ai) words to B register jk from (A0)
035ijk	,A0	Bjk,Ai	6-34	Memory	Store (Ai) words at B register jk to (A0)
†035ijk	0,A0	Bjk,Ai	6-34	Memory	Store (Ai) words at B register jk to (A0)
036ijk	Tjk,Ai	,A0	6-34	Memory	Read (Ai) words to T register jk from (A0)
†036ijk	Tjk,Ai	0,A0	6-34	Memory	Read (Ai) words to T register jk from (A0)
037ijk	,A0	Tjk,Ai	6-34	Memory	Store (Ai) words at T register jk to (A0)
†037ijk	0,A0	Tjk,Ai	6-34	Memory	Store (Ai) words at T register jk to (A0)
040ijkm	Si	exp	6-37	-	Transmit jkm to Si
041ijkm	Si	exp	6-37	-	Transmit exp = 1's complement of jkm to Si
042ijk	Si	<exp	6-38	S Logical	Form 1's mask exp = 64-jk bits in Si from the right
	Si	#>exp	6-38		
†042i77	Si	1	6-38	S Logical	Enter 1 into Si
†042i00	Si	-1	6-38	S Logical	Enter -1 into Si
043ijk	Si	>exp	6-38	S Logical	Form 1's mask exp = jk bits in Si from the left
	Si	#<exp			
†043i00	Si	0	6-38	S Logical	Clear Si
044ijk	Si	Sj&Sk	6-39	S Logical	Logical product of (Sj) and (Sk) to Si
†044ij0	Si	Sj&SB	6-39	S Logical	Sign bit of (Sj) to Si
†044ij0	Si	SB&Sj	6-39	S Logical	Sign bit of (Sj) to Si (j≠0)
045ijk	Si	#Sk&Sj	6-39	S Logical	Logical product of (Sj) and 1's complement of (Sk) to Si
†045ij0	Si	#SB&Sj	6-39	S Logical	(Sj) with sign bit cleared to Si
046ijk	Si	Sj\Sk	6-39	S Logical	Logical difference of (Sj) and (Sk) to Si
†046ij0	Si	Sj\SB	6-39	S Logical	Toggle sign bit of Sj, then enter into Si
†046ij0	Si	SB\Sj	6-39	S Logical	Toggle sign bit of Sj, then enter into Si (j≠0)
047ijk	Si	#Sj\Sk	6-39	S Logical	Logical equivalence of (Sk) and (Sj) to Si
†047i0k	Si	#Sk	6-39	S Logical	Transmit 1's complement of (Sk) to Si
†047ij0	Si	#Sj\SB	6-39	S Logical	Logical equivalence of (Sj) and sign bit to Si
†047ij0	Si	#SB\Sj	6-39	S Logical	Logical equivalence of (Sj) and sign bit to Si (j≠0)
†047i00	Si	#SB	6-39	S Logical	Enter 1's complement of sign bit into Si
050ijk	Si	Sj!Si&Sk	6-39	S Logical	Logical product of (Si) and (Sk) complement ORed with logical product of (Sj) and (Sk) to Si
†050ij0	Si	Sj!Si&SB	6-39	S Logical	Scalar merge of (Si) and sign bit of (Sj) to Si
051ijk	Si	Sj!Sk	6-39	S Logical	Logical sum of (Sj) and (Sk) to Si
†051i0k	Si	Sk	6-39	S Logical	Transmit (Sk) to Si
†051ij0	Si	Sj!SB	6-39	S Logical	Logical sum of (Sj) and sign bit to Si
†051ij0	Si	SB!Sj	6-39	S Logical	Logical sum of (Sj) and sign bit to Si (j≠0)
†051i00	Si	SB	6-39	S Logical	Enter sign bit into Si
052ijk	S0	Si<exp	6-43	S Shift	Shift (Si) left exp = jk places to S0
053ijk	S0	Si>exp	6-43	S Shift	Shift (Si) right exp = 64-jk places to S0
054ijk	Si	Si<exp	6-43	S Shift	Shift (Si) left exp = jk places
055ijk	Si	Si>exp	6-43	S Shift	Shift (Si) right exp = 64-jk places
056ijk	Si	Si,Sj<Ak	6-44	S Shift	Shift (Si and Sj) left (Ak) places to Si
†056ij0	Si	Si,Si<1	6-44	S Shift	Shift (Si and Sj) left one place to Si
†056i0k	Si	Si<Ak	6-44	S Shift	Shift (Si) left (Ak) places to Si

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
057ijk	Si	Sj,Si>Ak	6-44	S Shift	Shift (Sj and Si) right (Ak) places to Si
†057ij0	Si	Sj,Si>1	6-44	S Shift	Shift (Sj and Si) right one place to Si
†057i0k	Si	Si>Ak	6-44	S Shift	Shift (Si) right (Ak) places to Si
060ijk	Si	Sj+Sk	6-46	S Int Add	Integer sum of (Sj) and (Sk) to Si
061ijk	Si	Sj-Sk	6-46	S Int Add	Integer difference of (Sj) and (Sk) to Si
†061i0k	Si	-Sk	6-46	S Int Add	Transmit negative of (Sk) to Si
062ijk	Si	Sj+FSk	6-47	F.P. Add	Floating sum of (Sj) and (Sk) to Si
†062i0k	Si	+FSk	6-47	F.P. Add	Normalize (Sk) to Si
063ijk	Si	Sj-FSk	6-47	F.P. Add	Floating difference of (Sj) and (Sk) to Si
†063i0k	Si	-FSk	6-47	F.P. Add	Transmit normalized negative of (Sk) to Si
064ijk	Si	Sj*FSk	6-49	F.P. Mult	Floating product of (Sj) and (Sk) to Si
065ijk	Si	Sj*HSk	6-49	F.P. Mult	Half precision rounded floating product of (Sj) and (Sk) to Si
066ijk	Si	Sj*RSk	6-49	F.P. Mult	Full precision rounded floating product of (Sj) and (Sk) to Si
067ijk	Si	Sj*ISk	6-49	F.P. Mult	2 - Floating product of (Sj) and (Sk) to Si
070ijx	Si	/HSj	6-51	F.P. Rcpl	Floating reciprocal approximation of (Sj) to Si
071i0k	Si	Ak	6-52	-	Transmit (Ak) to Si with no sign extension
071i1k	Si	+Ak	6-52	-	Transmit (Ak) to Si with sign extension
071i2k	Si	+FAk	6-52	-	Transmit (Ak) to Si as unnormalized floating point number
071i3x	Si	0.6	6-52	-	Transmit constant 0.75*2**48 to Si
071i4x	Si	0.4	6-52	-	Transmit constant 0.5 to Si
071i5x	Si	1.	6-52	-	Transmit constant 1.0 to Si
071i6x	Si	2.	6-52	-	Transmit constant 2.0 to Si
071i7x	Si	4.	6-52	-	Transmit constant 4.0 to Si
072ixx	Si	RT	6-55	-	Transmit (RTC) to Si
073ixx	Si	VM	6-55	-	Transmit (VM) to Si
074ijk	Si	Tjk	6-55	-	Transmit (Tjk) to Si
075ijk	Tjk	Si	6-55	-	Transmit (Si) to Tjk
076ijk	Si	Vj,Ak	6-56	-	Transmit (Vj, element (Ak)) to Si
077ijk	Vi,Ak	Sj	6-56	-	Transmit (Sj) to Vi element (Ak)
†077i0k	Vi,Ak	0	6-56	-	Clear Vi element (Ak)
10hijkm	Ai	exp,Ah	6-57	Memory	Read from ((Ah) + exp) to Ai (A0=0)
†100ijkm	Ai	exp,0	6-57	Memory	Read from (exp) to Ai
†100ijkm	Ai	exp,	6-57	Memory	Read from (exp) to Ai
†10hi000	Ai	,Ah	6-57	Memory	Read from (Ah) to Ai
11hijkm	exp,Ah	Ai	6-57	Memory	Store (Ai) to (Ah) + exp (A0=0)
†110ijkm	exp,0	Ai	6-57	Memory	Store (Ai) to exp
†110ijkm	exp,	Ai	6-57	Memory	Store (Ai) to exp
†11hi000	,Ah	Ai	6-57	Memory	Store (Ai) to (Ah)
12hijkm	Si	exp,Ah	6-57	Memory	Read from ((Ah) + exp) to Si (A0=0)
†120ijkm	Si	exp,0	6-57	Memory	Read from (exp) to Si
†120ijkm	Si	exp,	6-57	Memory	Read from (exp) to Si
†12hi000	Si	,Ah	6-57	Memory	Read from (Ah) to Si
13hijkm	exp,Ah	Si	6-57	Memory	Store (Si) to (Ah) + exp (A0=0)
†130ijkm	exp,0	Si	6-57	Memory	Store (Si) to exp
†130ijkm	exp,	Si	6-57	Memory	Store (Si) to exp
†13hi000	,Ah	Si	6-57	Memory	Store (Si) to (Ah)
140ijk	Vi	Sj&Vk	6-59	V Logical	Logical products of (Sj) and (Vk) to Vi
141ijk	Vi	Vj&Vk	6-59	V Logical	Logical products of (Vj) and (Vk) to Vi
142ijk	Vi	Sj!Vk	6-59	V Logical	Logical sums of (Sj) and (Vk) to Vi
†142i0k	Vi	Vk	6-59	V Logical	Transmit (Vk) to Vi

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
143ijk	Vi	Vj!Vk	6-59	V Logical	Logical sums of (Vj) and (Vk) to Vi
144ijk	Vi	Sj\Vk	6-59	V Logical	Logical differences of (Sj) and (Vk) to Vi
145ijk	Vi	Vj\Vk	6-59	V Logical	Logical differences of (Vj) and (Vk) to Vi
†145iii	Vi	0	6-59	V Logical	Clear Vi
146ijk	Vi	Sj!Vk&VM	6-59	V Logical	Transmit (Sj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
†146i0k	Vi	#VM&Vk	6-59	V Logical	Vector merge of (Vk) and 0 to Vi
147ijk	Vi	Vj!Vk&VM	6-59	V Logical	Transmit (Vj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
150ijk	Vi	Vj<Ak	6-63	V Shift	Shift (Vj) left (Ak) places to Vi
†150ij0	Vi	Vj<1	6-63	V Shift	Shift (Vj) left one place to Vi
151ijk	Vi	Vj>Ak	6-63	V Shift	Shift (Vj) right (Ak) places to Vi
†151ij0	Vi	Vj>1	6-63	V Shift	Shift (Vj) right one place to Vi
152ijk	Vi	Vj,Vj<Ak	6-65	V Shift	Double shift (Vj) left (Ak) places to Vi
†152ij0	Vi	Vj,Vj<1	6-65	V Shift	Double shift (Vj) left one place to Vi
153ijk	Vi	Vj,Vj>Ak	6-65	V Shift	Double shift (Vj) right (Ak) places to Vi
†153ij0	Vi	Vj,Vj>1	6-65	V Shift	Double shift (Vj) right one place to Vi
154ijk	Vi	Sj+Vk	6-70	V Int Add	Integer sums of (Sj) and (Vk) to Vi
155ijk	Vi	Vj+Vk	6-70	V Int Add	Integer sums of (Vj) and (Vk) to Vi
156ijk	Vi	Sj-Vk	6-70	V Int Add	Integer differences of (Sj) and (Vk) to Vi
†156i0k	Vi	-Vk	6-70	V Int Add	Transmit negative of (Vk) to Vi
157ijk	Vi	Vj-Vk	6-70	V Int Add	Integer differences of (Vj) and (Vk) to Vi
160ijk	Vi	Sj*FVk	6-72	F.P. Mult	Floating products of (Sj) and (Vk) to Vi
161ijk	Vi	Vj*FVk	6-72	F.P. Mult	Floating products of (Vj) and (Vk) to Vi
162ijk	Vi	Sj*HVk	6-72	F.P. Mult	Half precision rounded floating products of (Sj) and (Vk) to Vi
163ijk	Vi	Vj*HVk	6-72	F.P. Mult	Half precision rounded floating products of (Vj) and (Vk) to Vi
164ijk	Vi	Sj*RVk	6-72	F.P. Mult	Rounded floating products of (Sj) and (Vk) to Vi
165ijk	Vi	Vj*RVk	6-72	F.P. Mult	Rounded floating products of (Vj) and (Vk) to Vi
166ijk	Vi	Sj*IVk	6-72	F.P. Mult	2 - floating products of (Sj) and (Vk) to Vi
167ijk	Vi	Vj*IVk	6-72	F.P. Mult	2 - floating products of (Vj) and (Vk) to Vi
170ijk	Vi	Sj+FVk	6-75	F.P. Add	Floating sums of (Sj) and (Vk) to Vi
†170i0k	Vi	+FVk	6-75	F.P. Add	Normalize (Vk) to Vi
171ijk	Vi	Vj+FVk	6-75	F.P. Add	Floating sums of (Vj) and (Vk) to Vi
172ijk	Vi	Sj-FVk	6-75	F.P. Add	Floating differences of (Sj) and (Vk) to Vi
†172i0k	Vi	-FVk	6-75	F.P. Add	Transmit normalized negatives of (Vk) to Vi
173ijk	Vi	Vj-FVk	6-75	F.P. Add	Floating differences of (Vj) and (Vk) to Vi
174ij0	Vi	/HVj	6-77	F.P. Rcpl	Floating reciprocal approximations of (Vj) to Vi
174ij1	Vi	PVj	6-78	F.P. Rcpl	Population counts of (Vj) to Vi
174ij2	Vi	QVj	6-78	F.P. Rcpl	Population count parities of (Vj) to Vi
175xj0	VM	Vj,Z	6-79	V Logical	VM=1 where (Vj) = 0
175xj1	VM	Vj,N	6-79	V Logical	VM=1 where (Vj) ≠ 0
175xj2	VM	Vj,P	6-79	V Logical	VM=1 where (Vj) positive
175xj3	VM	Vj,M	6-79	V Logical	VM=1 where (Vj) negative
176ixk	Vi	,A0,Ak	6-81	Memory	Read (VL) words to Vi from (A0) incremented by (Ak)
†176ix0	Vi	,A0,1	6-81	Memory	Read (VL) words to Vi from (A0) incremented by 1
177xjk	,A0,Ak	Vj	6-81	Memory	Store (VL) words from Vj to (A0) incremented by (Ak)
†177xj0	,A0,1	Vj	6-81	Memory	Store (VL) words from Vj to (A0) incremented by 1

† Special syntax form

I/O PROCESSOR INSTRUCTION SUMMARY E

IOP	APML	Description
000	PASS	No operation
001	EXIT	Exit from subroutine
002	I = 0	Disable system interrupts
003	I = 1	Enable system interrupts
004	A = A > d	Right shift C and A by d places, end off
005	A = A < d	Left shift C and A by d places, end off
006	A = A >> d	Right shift C and A by d places, circular
007	A = A << d	Left shift C and A by d places, circular
010	A = d	Transmit d to A
011	A = A & d	Logical product of A and d to A
012	A = A + d	Add d to A
013	A = A - d	Subtract d from A
014	A = k	Transmit k to A
015	A = A & k	Logical product of A and k to A
016	A = A + k	Add k to A
017	A = A - k	Subtract k from A
020	A = dd	Transmit operand register d to A
021	A = A & dd	Logical product of A and operand register d to A
022	A = A + dd	Add operand register d to A
023	A = A - dd	Subtract operand register d from A
024	dd = A	Transmit A to register d
025	dd = A + dd	Add operand register d to A, result to operand register d
026	dd = dd + 1	Transmit register d to A, add 1, result to operand register d
027	dd = dd - 1	Transmit register d to A, subtract 1, result to operand register d
030	A = (dd)	Transmit contents of memory addressed by register d to A
031	A = A & (dd)	Logical product of A and contents of memory addressed by register d, result to A
032	A = A + (dd)	Add contents of memory addressed by register d to A, result to A
033	A = A - (dd)	Subtract contents of memory addressed by register d from A, result to A

IOP	APML	Description
034	(dd) = A	Transmit A to memory addressed by register d
035	(dd) = A + (dd)	Add memory addressed by register d to A, result to same memory location
036	(dd) = (dd) + 1	Transmit memory addressed by register d to A, add 1, result to same memory location
037	(dd) = (dd) - 1	Transmit memory addressed by register d to A, subtract 1, result to same memory location
040	C = 1, iod = DN	Set carry equal to channel d done
041	C = 1, iod = BZ	Set carry equal to channel d busy
042	C = 1, IOB = DN	Set carry equal to channel B done
043	C = 1, IOB = BZ	Set carry equal to channel B busy
044	A = A > B	Right shift C and A by B places, end off
045	A = A < B	Left shift C and A by B places, end off
046	A = A >> B	Right shift C and A by B places, circular
047	A = A << B	Left shift C and A by B places, circular
050	A = B	Transmit B to A
051	A = A & B	Logical product of A and B to A
052	A = A + B	Add B to A, result to A
053	A = A - B	Subtract B from A, result to A
054	B = A	Transmit A to B
055	B = A + B	Add B to A, result to B
056	B = B + 1	Transmit B to A, add 1, result to B
057	B = B - 1	Transmit B to A, subtract 1, result to B
060	A = (B)	Transmit operand register B to A
061	A = A & (B)	Logical product of A and operand register B to A
062	A = A + (B)	Add operand register B to A, result to A
063	A = A - (B)	Subtract operand register B from A, result to A
064	(B) = A	Transmit A to operand register B
065	(B) = A + (B)	Add operand register B to A, result to operand register B
066	(B) = (B) + 1	Transmit operand register B to A, add 1, result to operand register B
067	(B) = (B) - 1	Transmit operand register B to A, subtract 1, result to operand register B
070	P = P + d	Jump to P + d
071	P = P - d	Jump to P - d
072	P = P + d	Return jump to P + d
073	P = P - d	Return jump to P - d
074	P = dd	Jump to address in operand register d
075	P = dd + k	Jump to sum of k and operand register d
076	R = dd	Return jump to address in operand register d
077	R = dd + k	Return jump to sum of k and operand register d

IOP	APML	Description
100	P = P + d, C = 0	Jump to P + d if carry = 0
101	P = P + d, C # 0	Jump to P + d if carry ≠ 0
102	P = P + d, A = 0	Jump to P + d if A = 0
103	P = P + d, A # 0	Jump to P + d if A ≠ 0
104	P = P - d, C = 0	Jump to P - d if carry = 0
105	P = P - d, C # 0	Jump to P - d if carry ≠ 0
106	P = P - d, A = 0	Jump to P - d if A = 0
107	P = P - d, A # 0	Jump to P - d if A ≠ 0
110	R = P + d, C = 0	Return jump to P + d if carry = 0
111	R = P + d, C # 0	Return jump to P + d if carry ≠ 0
112	R = P + d, A = 0	Return jump to P + d if A = 0
113	R = P + d, A # 0	Return jump to P + d if A ≠ 0
114	R = P - d, C = 0	Return jump to P - d if carry = 0
115	R = P - d, C # 0	Return jump to P - d if carry ≠ 0
116	R = P - d, A = 0	Return jump to P - d if A = 0
117	R = P - d, A # 0	Return jump to P - d if A ≠ 0
120	P = dd, C = 0	Jump to address in operand register d if carry = 0
121	P = dd, C # 0	Jump to address in operand register d if carry ≠ 0
122	P = dd, A = 0	Jump to address in operand register d if A = 0
123	P = dd, A # 0	Jump to address in operand register d if A ≠ 0
124	P = dd + k, C = 0	Jump to address in operand register d + k if carry = 0
125	P = dd + k, C # 0	Jump to address in operand register d + k if carry ≠ 0
126	P = dd + k, A = 0	Jump to address in operand register d + k if A = 0
127	P = dd + k, A # 0	Jump to address in operand register d + k if A ≠ 0
130	R = dd, C = 0	Return jump to address in operand register d if carry = 0
131	R = dd, C # 0	Return jump to address in operand register d if carry ≠ 0
132	R = dd, A = 0	Return jump to address in operand register d if A = 0
133	R = dd, A # 0	Return jump to address in operand register d if A ≠ 0

IOP	APML	Description
134	R = dd + k, C = 0	Return jump to address in operand register d + k if carry = 0
135	R = dd + k, C # 0	Return jump to address in operand register d + k if carry ≠ 0
136	R = dd + k, A = 0	Return jump to address in operand register d + k if A = 0
137	R = dd + k, A # 0	Return jump to address in operand register d + k if A ≠ 0
140	iod : 0	Channel d function 0
141	iod : 1	Channel d function 1
142	iod : 2	Channel d function 2
143	iod : 3	Channel d function 3
144	iod : 4	Channel d function 4
145	iod : 5	Channel d function 5
146	iod : 6	Channel d function 6
147	iod : 7	Channel d function 7
150	iod : 10	Channel d function 10
151	iod : 11	Channel d function 11
152	iod : 12	Channel d function 12
153	iod : 13	Channel d function 13
154	iod : 14	Channel d function 14
155	iod : 15	Channel d function 15
156	iod : 16	Channel d function 16
157	iod : 17	Channel d function 17
160	IOB : 0	Channel B function 0
161	IOB : 1	Channel B function 1
162	IOB : 2	Channel B function 2
163	IOB : 3	Channel B function 3
164	IOB : 4	Channel B function 4
165	IOB : 5	Channel B function 5
166	IOB : 6	Channel B function 6
167	IOB : 7	Channel B function 7
170	IOB : 10	Channel B function 10
171	IOB : 11	Channel B function 11
172	IOB : 12	Channel B function 12
173	IOB : 13	Channel B function 13
174	IOB : 14	Channel B function 14
175	IOB : 15	Channel B function 15
176	IOB : 16	Channel B function 16
177	IOB : 17	Channel B function 17

SYSTEM CHANNEL ASSIGNMENTS

F

The channel assignments for a typical Model 4400 system are shown in table F-1.

Table F-1. Typical Model 4400 system channel assignments

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Master I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXS	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Buffer I/O Processor
	7	AOA	Output to Buffer I/O Processor
	10	AIB	Input from Disk I/O Processor
	11	AOB	Output to Disk I/O Processor
	12	AIC	Input from Auxiliary I/O Processor
	13	AOC	Output to Auxiliary I/O Processor
	14		
	15		
	16	ERA	Error log
	17	EXB	Peripheral Expander (DMA 0)
	20	CIA	Input from CRAY-1 channel (DMA 1)
	21	COA	Output to CRAY-1 channel (DMA 1)
	22		
	23		
	24	CIB	Input from F.-E. Interface (DMA 2)
	25	COC	Output to F.-E. Interface (DMA 2)
	26		
	27		
	30	CIC	Input from F.-E. Interface (DMA 4)
	31	COC	Output to F.-E. Interface (DMA 4)
	32		
	33		
	34	CID	Input from F.-E. Interface (DMA 5)
	35	COD	Output to F.-E. Interface (DMA 5)
	36		
	37		
	40	TIA	Console 0 keyboard
	41	TOA	Console 0 display

Table F-1. Typical Model 4400 system channel assignments (continued)

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Master I/O Processor (continued)	42	TIB	Console 1 keyboard
	43	TOB	Console 1 display
	44		
	45		
	46		
	47		
Buffer I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXS	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Master I/O Processor
	7	AOA	Output from Master I/O Processor
	10	AIB	Input from Disk I/O Processor
	11	AOB	Output to Disk I/O Processor
	12	AIC	Input from Auxiliary I/O Processor
	13	AOC	Output to Auxiliary I/O Processor
	14	HIA	Input from Memory Channel (DMA 4)
	15	HOA	Output to Memory Channel (DMA 4)
	16		
	17		
	20	DKA	Disk Storage Unit 0 (DMA 0)
	21	DKB	Disk Storage Unit 1 (DMA 0)
	22	DKC	Disk Storage Unit 2 (DMA 1)
	23	DKD	Disk Storage Unit 3 (DMA 1)
	24	DKE	Disk Storage Unit 4 (DMA 2)
	25	DKF	Disk Storage Unit 5 (DMA 2)
	26	DKG	Disk Storage Unit 6 (DMA 5)
	27	DKH	Disk Storage Unit 7 (DMA 5)
	30		
	31		
	32		
	33		
	34		
	35		
	36		
	37		
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		

Table F-1. Typical Model 4400 system channel assignments (continued)

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Disk I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXR	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Master I/O Processor
	7	AOA	Output to Master I/O Processor
	10	AIB	Input from Buffer I/O Processor
	11	AOB	Output to Buffer I/O Processor
	12	AIC	Input from Auxiliary I/O Processor
	13	AOC	Output to Auxiliary I/O Processor
	14		
	15		
	16		
	17		
	20	DKA	Disk Storage Unit 0 (DMA 0)
	21	DKB	Disk Storage Unit 1 (DMA 0)
	22	DKC	Disk Storage Unit 2 (DMA 0)
	23	DKD	Disk Storage Unit 3 (DMA 0)
	24	DKE	Disk Storage Unit 4 (DMA 1)
	25	DKF	Disk Storage Unit 5 (DMA 1)
	26	DKG	Disk Storage Unit 6 (DMA 1)
	27	DKH	Disk Storage Unit 7 (DMA 1)
	30	DKI	Disk Storage Unit 8 (DMA 2)
	31	DKJ	Disk Storage Unit 9 (DMA 2)
	32	DKK	Disk Storage Unit 10 (DMA 2)
	33	DKL	Disk Storage Unit 11 (DMA 2)
	34	DKM	Disk Storage Unit 12 (DMA 5)
	35	DKN	Disk Storage Unit 13 (DMA 5)
	36	DKO	Disk Storage Unit 14 (DMA 5)
	37	DKP	Disk Storage Unit 15 (DMA 5)
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		

Table F-1. Typical Model 4400 system channel assignments (continued)

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Auxiliary I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXS	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Master I/O Processor
	7	AOA	Output to Master I/O Processor
	10	AIB	Input from Buffer I/O Processor
	11	AOB	Output to Buffer I/O Processor
	12	AIC	Input from Disk I/O Processor
	13	AOC	Output to Disk I/O Processor
	14		
	15		
	16		
	17		
	20	BMA	Block Multiplexer Channel 0 (DMA 0)
	21	BMB	Block Multiplexer Channel 1 (DMA 0)
	22	BMC	Block Multiplexer Channel 2 (DMA 0)
	23	BMD	Block Multiplexer Channel 3 (DMA 0)
	24	BME	Block Multiplexer Channel 4 (DMA 1)
	25	BMF	Block Multiplexer Channel 5 (DMA 1)
	26	BMG	Block Multiplexer Channel 6 (DMA 1)
	27	BMH	Block Multiplexer Channel 7 (DMA 1)
	30	BMI	Block Multiplexer Channel 8 (DMA 2)
	31	BMJ	Block Multiplexer Channel 9 (DMA 2)
	32	BMK	Block Multiplexer Channel 10 (DMA 2)
	33	BML	Block Multiplexer Channel 11 (DMA 2)
	34	BMM	Block Multiplexer Channel 12 (DMA 5)
	35	BMN	Block Multiplexer Channel 13 (DMA 5)
	36	BMO	Block Multiplexer Channel 14 (DMA 5)
	37	BMP	Block Multiplexer Channel 15 (DMA 5)
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		

IOP PROGRAMMING CONSIDERATIONS G

Several special cases must be considered when programming the I/O Processor. These cases are explained below.

EXIT STACK TIMING

When issuing PXS : 4 or PXS : 15, allow 4 CPs[§] before enabling system interrupts, return jumps or exit instructions. PXS : 14 and PXS : 15 instructions should only be used when system interrupts are disabled.

EXIT STACK INTERRUPT HANDLING

If return jumps are used in an interrupt handler, verifiy that enough levels are left available in the stack. An interrupt with the exit stack pointer at 13_{10} causes the pointer to go to 14_{10} and leave only one location open. A worse case exists if a return jump which causes a program fetch request (PFR) interrupt is issued with the stack pointer at 13_{10}. The return address goes in 14_{10} and the interrupt address goes into 15_{10}. This condition now leaves two interrupts present--both the exit stack boundary and PFR, with the PFR being the highest priority and no stack locations available. If the stack pointer is allowed to increment from 15_{10}, it clears to 0, and incorrect return addresses are used.

SYSTEM INTERRUPT ENABLE

When issuing an I=1, the system interrupt enable is delayed until the next non-branch or non-I/O instruction is issued. The instructions that do not enable interrupts are the 40-43 and 70-137.

---

§ The term clock periods refers to processor instruction times taken up by issuing pass instructions, or some other instruction or group of instructions whose execution time equals or exceeds the delays noted. Any instruction or group of instructions may be used as long as they are not included in any of the special cases stated in this list.

This allows the executive/monitor to get back to the interruptible activity before an interrupt is accepted.

Use an I=0 instruction at the interrupt handler entrance. If a redundant I=1 is executed and an interrupt occurs before a non-branch or non-I/O instruction is encountered, the interrupt handler is entered (with interrupts disabled). But interrupts are re-enabled when the first non-branch or non-I/O instruction is issued within the interrupt handler.

SYSTEM INTERRUPT DISABLE

The instructions following an I=0 instruction may be skipped if an interrupt occurs (while I=0 is executing). Hence, a pass instruction should follow every I=0.

SYSTEM INTERRUPT CLEARED OR SET BY THE ENABLES FOR INDIVIDUAL CHANNELS

After issuing a command 6 or 7 to any I/O channel, allow 3 CPs[§] before seeing its effect on system interrupt. (Assuming system interrupts are, or will be, enabled.)

I/O CHANNEL TIMING

When issuing any command to the I/O channels, allow 1 CP[§] before checking busy or done status.

Also allow 1 CP[§] after any command 6 or 7 before checking for interrupt number (IOR : 10).

BUFFER MEMORY ERRORS

If a Buffer Memory multiple bit error has occurred, an MOS : 0 instruction is required prior to the next read or write command. The interface operation waits indefinitely on the error if it is not cleared by the MOS : 0.

---

§ The term clock periods refers to processor instruction times taken up by issuing pass instructions, or some other instruction or group of instructions whose execution time equals or exceeds the delays noted. Any instruction or group of instructions may be used as long as they are not included in any of the special cases stated in this list.

BUFFER MEMORY DEADSTART TIME

Deadstarting a processor via Buffer Memory requires apporoximately 2 ms to transfer the full 64K into I/O memory. This time assumes no other Buffer Memory activity other than refresh.

ERROR LOGGING AND BLOCK MULTIPLEXER CHANNELS

The commands : 10-13 to the error logging channel or the block multiplexer channel decode the present accumulator data on the interface. If the IOP instruction previous to the commands : 10-13 is any of the following: 4-7, 12, 13, 16, 17, 22, 23, 32, 33, 44-47, 52, 53, 62, 63; then an 11 or 15 instruction (with the d or k field set to all ones) should be inserted between the instruction and the interface command. This avoids a 1-CP timing restriction caused by the adder and shifter.

I/O INSTRUCTIONS AFTER DEADSTART

The first instruction executed after a deadstart cannot be an I/O instruction. This includes 40-43, 140-147, 154-157, 160-167, 174-177. The accumulator must be loaded before executing any of these instructions. This avoids a special control sequence condition after deadstart.

NOTE

150-153, 170-173 instructions may be executed after deadstart as the instructions do load the accumulator.

PERIPHERAL EXPANDER CHANNEL TRANSFERS

The expander channel supports block transfers to only the first $100{,}000_8$ parcels of I/O memory.

LIST OF ABBREVIATIONS

A, An	CPU address register n, n = 0 to 7 (CPU); IOP accumulator
Addr	Address
Adv.	Advance
Ai,Aj,Ak	Address register specified by instruction i,j,k fields
APML	A Programming Machine Language
B, Bn	CPU intermediate address register n, n = 0 to 77_8; IOP B register
BA	Bank address, buffer address
BIOP	Buffer I/O Processor
Bjk	Buffer register specified by instruction j,k fields
BM	Buffer Memory
CA	Current address register
CAL	Cray Assembly Language
Ch	Channel
CL	Channel Limit register
CIP	Current Instruction Parcel register
CLK	Clock
Contr.	Control
CP	Clock period, central processor
CPU	Central processing unit
CRI	Cray Research, Incorporated
DCU	Disk controller unit
Distr.	Distribution
DIOP	Disk I/O Processor
DMA	Direct memory access
DP	Destination Pointer register
DSU	Disk Storage Unit
Exch.	Exchange
F	Flag register (exchange package); IOP instruction function field

F.E.	Front end
F.P.	Floating-point
F.U.	Functional Unit
FWA	First word address
gh	g and h fields, CPU instruction operation code
GR	Group
h	h field, CPU instruction
Hz	Hertz, cycles per second
i	i field of CPU instruction
II	Interrupt Interval register (CPU); Instruction Issue register (IOP)
IC	Integrated Circuit
ICD	Interrupt Countdown counter
I/O	Input/Output
IOP	I/O Processor
IOR	I/O request
k	Kilo, 1024, k field CPU instruction
jk	j and k fields, CPU instruction
LA	Limit address
LIP	Last instruction parcel
LSI	Large scale integration
LWA	Last word address
M	Million; mode bit field in exchange package; instruction field
MBits	Megabits or million bits
Mbyte	Megabyte or million bytes
MCU	Maintenance Control Unit
MG	Motor-Generator
MHz	MegaHertz, or million cycles per second
MIOP	Master I/O Processor
MMI	Monitor Mode Interrupt
MOS	Metal oxide semiconductor
MSEC	Millisecond
MS	Mass storage
MSKO	Mask out
NIP	Next instruction parcel

nmos	Negative channel metal oxide semiconductor
ns	Nanosecond
OS	Operating system
Osc	Oscillator
P	Program address register; Program parcel counter
PDU	Power Distribution Unit
PFR	Program fetch request
POP	Population count
PCI	Programmable clock interrupt
R	Request; response
RA	Read address
Recip	Reciprocal
Reg.	Register
Req.	Request
Resp.	Response
Ref.	Reference
RP	Register Pointer register
R'RAB	R' = high-order bits of read address, RA = Read Address, B = Bank low-order bits of address in exchange package
RTC	Real-time clock
S	Scalar
s	Second
Seq.	Sequence
Sn	Scalar register, n = 0 to 7
Si, Sj, Sk	Scalar register specified by instruction i or j or k field
SECDED	Single error correction/double error detection
Stor.	Storage
T, Tn	Intermediate scalar register n, n = 0 to 77_8
Tjk	Temporary register indicated by instruction j,k fields
V, Vn	Vector register n, n = 0 to 7
Vi, Vj, Vk	Vector register specified by instruction i or j or k field
VL	Vector Length register
VM	Vector Mask register
XA	Exchange Address register
XIOP	Auxiliary I/O Processor

READERS COMMENT FORM

CRAY-1 S Series Hardware Reference Manual HR-0808 B

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.

CUT ALONG THIS LINE

FOLD

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

Hwy 178 North
Chippewa Falls, WI 54729
U.S.A.

FOLD

STAPLE

Cray Research, Inc.
Publications Department
1440 Northland Drive
Mendota Heights, MN 55120
612-452-6650
TLX 298444

UCID-30179
REVISION 2
COMPUTER
DOCUMENTATION

LAWRENCE LIVERMORE LABORATORY
University of California / Livermore, California

TIMING CODES ON THE CRAY-1: PRINCIPLES AND APPLICATIONS

Harry L. Nelson

May 10, 1981

Prepared for U.S. Department of Energy under Contract No. W-7405-Eng-48

AVAILABILITY

This document is available online as follows:

```
XPORT RD .717675:UCID:UCID30173 / 1 1
```

View the print file on the TMDS, or print it as follows:

```
TRIX AC / 1 1
.PRINT(<NIP UCID30179 BOX ann identification>)
.END
```

CONTENTS

	Page
Abstract	1
I. Introduction	1
II. Overall Time Analysis on the CRAY-1	2
Code Timing with TIMER and TALLY	2
FLOWTRACE	6
Call Second(0)	7
IRTC and/or Q8RTC	10
Other Methods	13
III. Predicting Timing	14
General Remarks	14
The Basic Details	22
Two Short Examples	23
The Timing Chart	24
Preliminary Considerations	24
Two Basic Examples and Comments	26
Example 1	26
Example 2	31
Conclusions	34
IV. The Computer Code CYCLES	35
Cycles Writeup	35
Abilities and limitations	36
VL and A registers	36
Jump instructions	37
Instruction buffer (I-buff) delays	37
Availability of CYCLES	38
Table of Delay Codes	40
V. Examples	41
ZVSEEK	41
Main Loop of ZVSEEK (Old Version)	41
Improved Version with XOR Replaced by Fixed Subtract	42
QVDIVO	46
QVSQRTH	49
Appendix A. An Abridgement of the Summary of CPU Timing Information	52
SCALAR INSTRUCTIONS	52
Vector Instructions	53
Appendix B. What Happens When You Run on the CRAY?	56
Appendix C. The details of Instruction Fetch Timing	58
Summary	62

TIMING CODES ON THE CRAY-1: PRINCIPLES AND APPLICATIONS

ABSTRACT

Complete instruction-timing information for the CRAY-1 computer is presented together with a method of recording the minimum necessary details for precise prediction of the running time of various algorithms. Several examples of optimum assembly language coding are listed, with comments that illustrate the timing details. Usage of the code CYCLES which predicts timing of actual CAL, CFT, or CIVIC programs is described. Usage of codes TIMER and TALLY is described.

I. INTRODUCTION

The aim of this document is to show how to locate and analyze the segments of a code that are important from a timing viewpoint. Computer codes TIMER and TALLY are useful for this purpose. Then, having identified critical sections, we consider how to perform them optimally. Computer code CYCLES is of value in obtaining such performance.

On the CRAY-1, optimum programming consists of finding the best algorithm and avoiding conflicts in implementing it. Usually the best algorithm can be characterized as a "parallel vector" algorithm.

Once an algorithm has been decided upon, one must consider how it can be implemented with actual hardware instructions. The algorithm may have to be changed if it causes unavoidable conflicts due to the shared nature of the CRAY-1's data paths, registers, functional units, and memory. Avoiding conflicts is primarily a matter of understanding the timing details involved.

Several examples of improved performance achieved through timing analysis will be given. (For a description of the environment at LLNL in which your code will run, see Appendix B.)

II. OVERALL TIME ANALYSIS ON THE CRAY-1

The first step in improving the performance of a code is to find out where it is spending its time. In most programs there is some small iterative algorithm that uses the majority of the CPU time. Thus, improvements to a very limited number of lines of code can result in dramatic reductions in the amount of time required to perform a calculation. In particular, if you have a FORTRAN program in which, say, 70% of the time is spent in one inner DO loop, you can limit your effort, initially, to making improvements to that loop. In such cases, obviously, the use of assembly language should be considered. Much of this report will be concerned with time analysis of relatively small assembly language routines. However, initially we look at full code analysis.

Code Timing with TIMER and TALLY

The LASNEX code group, primarily Jim Kohn and George Zimmerman, has put together a simple set of tools to do code timing on the CRAY-1 (and 7600). The capabilities are similar to BEGINMAP-ENDMAP but are simpler to use. The output produced by this set of tools is much less extensive than BEGINMAP but contains the essential ingredients to do timing analysis for almost any code.

Timer

TIMER is a subroutine which you must call in your code. The call looks like:

```
CALL TIMER(IOC,'FNAME',BUFFER,LBUFFER,'HEADER',LHEADER)
```

where,

IOC — is an I/O Connector (IOC) available for I/O. However, if this IOC ever becomes unavailable, TIMER tries to find another one. The IOC is active only during actual writes to disk by TIMER. IOC=0 is satisfactory.

FNAME — is a file sequence name. A sequenced name is formed from this by appending a digit (usually 0) on the right end of the name truncating the leftmost character if necessary. If FNAME already ends with a decimal digit, FNAME is used as is for the first file in the sequence. If any file in the sequence already exists it will be destroyed.

BUFFER — is an I/O buffer. It must be permanently available and reserved for TIMER's use only. Otherwise garbage could be written to disk.

LBUFFER is the length of the I/O buffer. It may be any size convenient for the user. 512 words seems to work quite well.

HEADER is an ASCII string which will be written into the beginning of the disk file to identify this timing file (in case multiple runs are made). Date, time, code name, problem name are some possible items that you may wish to put in the header.

LHEADER is the word length of HEADER. It must be at least one word long, even if the header itself is blank.

TIMER operates by interrupting your code every 4 milliseconds and finding out what the p-counter is. It stores the p-counter in the buffer, dumps the buffer if necessary, and then returns from interrupt. TIMER itself does not perform any actual timing analysis. It just creates a timing file with p-counters in it. The actual analysis is done by the TALLY code.

To obtain a complete timing analysis of your code, TIMER should be called as early as possible during the execution of your code. Once the call to TIMER has been made, no other calls are required until your code wants to terminate the timing analysis. Your code should not be affected by the presence of TIMER in it. The overhead is approximately 5 microseconds per interrupt, which should not be detectable. TIMER contains only about 100 lines of FORTRAN so it is very small.

To terminate the timing analysis, a call must be made to TIMEND. TIMEND is called with no arguments. It shuts down the timing, flushes the buffer, closes the file and truncates it. TIMEND is an entry point inside TIMER.

No externals are required by TIMER (or TIMEND). It is self-contained. It is available by loading your code with ALIBCRAY. If you cannot access ALIBCRAY, the source for TIMER may be extracted from file CLASS, and compiled to produce a binary file for LDR.

TIMER stores one other piece of information in the timing file along with the p-counter. This is a process index. This index is read from common block /Q8LDBKX/ which is one word long. By default this word is set to 1. Your code may set this word at any time to designate the current process which is active. The only reason to do this would be to obtain a more detailed breakdown of the usage of utility subroutines (e.g., SQRT, LOG, EXP, BASELIB routines, etc.) according to the structure of your code. For example, you could find out which logical process in your code is using SQRT the most. This feature is usually used in overlayed (or segmented) codes where the overlay (or segment) number can be stored into this common block. But any single level code could use this equally well. Maximum value for this process index is 255 on CRAY.

Tally

The TALLY code requires 2 files in order to do a timing analysis. The first is the set of timing files (usually 1 file) produced by the TIMER routine. The second file is the symbol table file produced by the loader. The symbol table is usually contained in your controllee file so you may normally use your executing code name as the symbol table file. A copy of TALLY can be extracted from public file "NELSON", at LLNL.

The execute line to run TALLY is:

TALLY timing-file-name symbol-table-file [options] / t v

where the following options are available,

none (i.e., no options specified). This does a short timing analysis. Histograms on a subroutine by subroutine basis are not produced.

ALL. This does a complete timing analysis producing all of the output TALLY can. Most people use this option.

BS. n Set the Bin Size to n parcels. Tally accumulates timing information into bins. Each bin represents n parcels of your code. Default is n=32 (8 words) which works very nicely.

The timing analysis produced by TALLY is fairly straightforward to understand. It is broken into 3 logical sections. Each sections includes percentage breakdowns as well as actual numbers of hits. The term "hit" designates an instance of the p-counter being in a given routine or a given bin.

The first section does on overall timing analysis. The number of hits in each subprogram as well as the percent of the total time the subprogram used is listed. A subprogram appears in this list only if at least 1 hit was recorded within its bounds.

The second section does a similar kind of analysis but by process index. Thus this is a bit more detailed. The usage of commonly used utility subprograms is broken up by process index.

The third section (if requested with ALL.) is a detailed analysis (via histogram) of each subprogram for which hits were recorded. The breakdown is by bins where a bin represents a small section of code. The number of hits within a bin is printed along with a 'bar' indicating graphically the relative time spent within the bin. Note that the algorithm determining the length of the 'bar' is non-linear. The actual hit count must be used for an accurate, detailed analysis.

Example of output from TALLY.

First, for a GRAFLIB 'typical' test problem written to identify those routines in which time was being spent.

01/27/81

NHIT= 998

LOCATION	LENGTH	SUBROUTINE	NHIT	PERCENT
00061626	00000635	MAIN.	3	.3006
00063560	00000015	RNFL	3	.3006
00064764	00000515	JPPL2A	1	.1002
00067425	00000106	ZMOVEBIT	7	.7014
00071430	00000634	KXDRPL	2	.2004
00072275	00000040	ZMOVEWRD	3	.3006
00073725	00000146	KXVT2D	126	12.6253
00074073	00000230	KXCL2D	444	44.4890
00075740	00002450	KPFRLN	356	35.6713
00110614	00000070	QBPAK	51	5.1102
00113633	00000041	IZIOSTAT	2	.2004

Second, after about one personal month of effort spent recording the three main time-consuming routines into CALL.

03/16/81

NHIT= 218

LOCATION	LENGTH	SUBROUTINE	NHIT	PERCENT
00061653	00000635	MAIN.	2	.9174
00063605	00000015	RNFL	4	1.8349
00063622	00000121	ZCITOA	1	.4587
00065276	00000626	JPPL2A	1	.4587
00070173	00000106	ZMOVEBIT	4	1.8349
00074424	00000620	KXDRPL	1	.4587
00075255	00000040	ZMOVEWRD	4	1.8349
00077130	00000040	HKXVT2D	7	3.2110
00077170	00000076	HCL2D	128	58.7156
00101335	00002410	KPFRLN	6	2.7523
00114365	00000525	KFRVEC	10	4.5872
00115112	00000070	QBPAK	49	22.4771
00115202	00000060	KWBFFN	1	.4587

Another month spent developing and coding vector versions of HCL2D and QBPAK reduced them to 34 and 19 hits respectively, and resulted in a final tenfold improvement for this heavily used LLNL utility, (NHIT= 97).

FLOWTRACE

Often one would like to find out which subroutines of a large code are frequently called and gain an overall knowledge of its flow. CFT users can accomplish this by using FLOWTRACE. This is a compile-time option, which, although expensive, does produce a rather nice breakdown of a code's behavior.

An example of the output from FLOWTRACE is shown below. Full details and assistance are available from the local CRAY representatives.

```
    ROUTINE              TIME      %   CALLED  AVERAGE T
 1  FENBTV           0.059817   1.18        1  0.059817
                                                    CALLS  THGEN
 2  THGEN            0.067451   1.33       23  0.002933  CALLED BY  FENBTV
 3  BCOND            0.034805   0.68        1  0.034805  CALLED BY  FENBTV
 4  ICOND            0.023386   0.46        1  0.023386  CALLED BY  FENBTV
 5  PREFRON          0.001754   0.03        1  0.001754  CALLED BY  FENBTV
 6  VSTRAP           0.087725   1.72        1  0.087725  CALLED BY  FENBTV
                                                    CALLS  OUTSOL
 7  OUTSOL           1.190628  23.41       46  0.025883  CALLED BY  VSTRAP
 8  FRONT            1.455038  28.60       22  0.066138  CALLED BY  VSTRAP
                                                    CALLS  QVSET
 9  QVSET            0.010048   0.20       94  0.000107  CALLED BY  FRONT
10  MAKEL            0.035852   0.70        6  0.005975  CALLED BY  FRONT
                                                    CALLS  QVSET
11  BASIS            0.000900   0.02        9  0.000100  CALLED BY  MAKEL
12  MAKEQ            0.226627   4.45      132  0.001717  CALLED BY  FRONT
                                                    CALLS  NLMAT
13  NLMAT            0.117950   2.32      132  0.000894  CALLED BY  MAKEQ
                                                    CALLS  ENCOM
14  ENCOM            0.017083   0.34      132  0.000129  CALLED BY  NLMAT
15  NLRHS            0.001008   0.02        6  0.000168  CALLED BY  MAKEQ
16  BACSUB           0.679884  13.37       22  0.030904  CALLED BY  FRONT
17  ITER             0.058392   1.15       21  0.002781  CALLED BY  VSTRAP
                                                    CALLS  QVSET

***     TOTAL        5.087028
***  OVERHEAD        0.033296

SUBROUTINE LINKAGE OVERHEAD SUMMARY                  922 CALLS

                 MINIMUM    MAXIMUM    AVERAGE     CYCLES     SECONDS        %
T REGISTERS            0         22        6.2      28594    3.57e-04   0.0070
B REGISTERS            2          8        4.3      26306    3.29e-04   0.0065
  ARGUMENTS            0          5        0.8       2876    3.60e-05   0.0007
      total                                         57776    7.22e-04   0.0142
 MAXIMUM SUBROUTINE DEPTH = 7
```

Call Second(0)

Gathering timing information can be made an integral part of a routine. A basic tool I recommend for this use within a specific FORTRAN subroutine is the FORTLIB function SECOND. On the CRAY-1, SECOND returns the total unweighted CPU time charged against your code since execution began. Calls to SECOND are relatively cheap (approximately 5 microseconds per call) and are not subject to variations due to the current time-sharing load on the machine. Other techniques may be used for finer analysis of small code sections, but for overall purposes SECOND is adequate. An example of its use is shown in the code below.

```
      PROGRAM MF301T(UNIT59=TTY)
      COMMON D(1325)
      DIMENSION (1024)
      CALL LINK('UNIT59=TERMINAL//')
      E = SECOND(0)
      TM = SECOND(0)-E
      TT = TM*976.*25.*4.
      T5 = 0
      T2 = 0
      X = .125
      Y = .015625
      A = 15.5
      WRITE(59,58) A,X,Y
58    FORMAT('CHECKING FOR A = ',F7.4,'  X =  ',F7.5,'  Y =  ',F8.6)
      DO 4 K = 1,25
      B = A+X*K
      DO 1 M=1,1325
1     D(M) = B*B-M
      DO 3 J = 1,976
      C = Y*J
      TA = SECOND(0)
      DO 5 I=1,1024
      F(I) = (C-B*D(I))/2.
5     CONTINUE
      TB = SECOND(0)
      T5 = T5+TB-TA-TM
      TA = SECOND(0)
      DO 2 I=1,1024
      IF(F(I).NE.0) GO TO 2
      E = SECOND(0)
      WRITE(59,60) B,C,D(I),E,I,J,K
60    FORMAT('HIT AT',4F9.4,3I5)
2     CONTINUE
      TB = SECOND(0)
      T2 = T2+TB-TA-TM
3     CONTINUE
4     CONTINUE
```

```
      E = SECOND(0)
      WRITE(59,59) A,E,I,J,K
      WRITE(59,61) T5,T2,TT
61    FORMAT('LOOP5 TIME =',F9.4,3X,'LOOP2 TIME =',F9.4,3X
     % ,'CLOCK CALL TIME =',F9.4)
      STOP 1
59    FORMAT('            A       TIME      I     J     K',/,'NO HIT',
     % 2F9.4,3I5)
      END
```

```
+-------------------------------------------------------------------+
|                                                                   |
|Note:  The source code for this example, MF301T, as well as the    |
|sources for all other examples in this writeup are resident on     |
|the CRAY-1 in public LIB file CLASS.  One can extract and run      |
|this example using the CIVIC compiler as follows (lower case       |
|typing represents user input; upper case is computer output):      |
+-------------------------------------------------------------------+
```

```
lib class
C 06/13/79 09:41:03 644400
OK. x mf301t
OK. end

 ALL DONE
civic mf301t mfc
*** CRAY LOADER VERSION - C120  03/08/79

 ALL DONE
mfc
CHECKING FOR A =   15.5000    X =   0.12500    Y =   0.015625
HIT AT   15.7500    0.9844    0.0625    1.6408   248    63     2
HIT AT   16.2500    1.0156    0.0625    7.8015   264    65     6
HIT AT   16.5000    4.1250    0.2500   11.1936   272   264     8
HIT AT   16.7500    9.4219    0.5625   14.8103   280   603    10
HIT AT   17.2500    9.7031    0.5625   20.9958   297   621    14
HIT AT   17.5000    4.3750    0.2500   23.5364   306   280    16
HIT AT   17.7500    1.1094    0.0625   26.2874   315    71    18
HIT AT   18.2500    1.1406    0.0625   32.4474   333    73    22
HIT AT   18.5000    4.6250    0.2500   35.8796   342   296    24
            A       TIME      J     J     K
NO HIT   15.500   38.4962 1025   977    26
LOOP5 TIME =   15.6641    LOOP2 TIME =   18.4826    CLOCK CALL TIME =   4.2944
```

The following, for comparison, is the CFT version, which is automatically vectorized for loop 5:

```
  rcft i=mf301t,go
  CF000 - CFT VERSION -    01/23/81 1.09b
  CF001 - COMPILE TIME =      0.0346 SECONDS
  CF002 -         54 LINES,          44 STATEMENTS
  *** CRAY LOADER VERSION - C120  03/08/79
  CHECKING FOR A =   15.5000    X =  0.12500    Y =  0.015625
  HIT AT   15.7500    0.9844    0.0625    0.9592  248    63     2
  HIT AT   16.2500    1.0156    0.0625    4.5498  264    65     6
  HIT AT   16.5000    4.1250    0.2500    6.5256  272   264     8
  HIT AT   16.7500    9.4219    0.5625    8.6349  280   603    10
  HIT AT   17.2500    9.7031    0.5625   12.2467  297   621    14
  HIT AT   17.5000    4.3750    0.2500   13.7301  306   280    16
  HIT AT   17.7500    1.1094    0.0625   15.3368  315    71    18
  HIT AT   18.2500    1.1406    0.0625   19.9301  333    73    22
  HIT AT   18.5000    4.6250    0.2500   20.9280  342   296    24
           A        TIME      I    J     K
  NO HIT   15.5000  22.4518 1025   977    26
  LOOP5 TIME =    1.1869    LOOP2 TIME =   16.9523    CLOCK CALL TIME =   4.1968
```

From these numbers, we can see that (for the CFT version, at least) improvement efforts should be directed toward loop 2. (And, of course, the calls to SECOND will be eventually removed.)

IRTC and/or Q8RTC

The CRAY-1 has a cycle counter as one of its hardware features. This is a counter which steps by one each machine clock period of 12.5 nanoseconds. Detailed timing of code sections can be done using this counter. However, the counter steps whether or not your program is running, so care must be taken with its use in the time-sharing environment. The counter, called RTC (for real-time clock), is directly readable using FORTRAN. With CFT, one uses the construct. N = IRTC(0), and with CIVIC, N = Q8RTC(0), where N is an integer variable name. The compiler generates only the code necessary for reading the RTC and storing the reading in memory location N, a total of 48 bits of code, normally requiring only 3 extra clock periods to perform. (In certain cases a longer time is required because of an S-register, path, or memory conflict.)

The use of IRTC is illustrated in the session below. In the example, a FORTRAN routine calls a CAL assembly routine, which adds the first 51 elements of arrays A and B and places the result into array C by use of a scalar loop.

Here, it was possible to improve the performance of the machine on this example by about 6% by merely reordering the modules in memory. There are (admittedly pathological) examples of this type of thing where a change in running time of 100% occurs. Such changes are due to the avoidance of (or introduction of) conflicts.

First, the source codes for the example are extracted.

```
lib class
C 07/06/79 13:19:54 644400
OK. x abcs abcsf
OK. end

 ALL DONE
```

```
trixgl o!abcs
     19 LINES (   80S)
.t
 1 *        CAL  I=ABCS,B=BABCS,L=LSC
 2              IDENT              ABCS
 3              COMMON           ABCOMMON
 4 A            BSS                57
 5 B            BSS                56
 6 C            BSS                56
 7              BLOCK              ABCS
 8              ENTRY              ABCS
 9 ABCS          A1                 0
10              A2                 51
11  LOOP        S1                 A,A1
12              S2                 B,A1
13              S3                 S1+FS2
14              C,A1               S3
15              A1                 A1+1
16              A0                 A1-A2
17              JAN                LOOP
18              J                  BOO
19              END
.run
CAL  I=ABCS,B=BABCS,L=LSC
%PC3
[3.000]
CA012 - 0062K MEMORY + 0117K I/O BUFFERS USED

 ALL DONE


o!abcsf
     17 LINES (   80S)
.t
 1 *       CFT  I=ABCSF,ON=0,L=LSF,B=BSF
 2 *         LDR  I=(BSF,BABCS),ML=MSF,X=XBS,ORDER=CLNB,FIRST=BSF
 3 *      XBS
 4           COMMON /ABCOMMON/ A(56),OUTRANGE,B(56),C(56)
 5           CALL LINK('UNIT59=TERMINAL//')
 6           Y = X*X*X*X*X*X*Y*X
 7           DO 1 I = 1,169
 8    1      A(I) = I
 9           OUTRANGE = 6000040000000000000000B
10           M = IRTC(0)
11           CALL ABCS
12           N = IRTC(0)
13           X = N-M
14           WRITE(59,59) C,X
15   59      FORMAT(7F6.0)
16           STOP
17           END
```

```
.run
CFT I=ABCSF,ON=G,L=LSF,B=BSF
FT004 - CFT VERSION -    04/06/79 SCHEDULER
FT001 - COMPILE TIME =       0.0195 SECONDS

 ALL DONE
LDR I=(BSF,BABCS),ML=MSF,X=XBS,ORDER=CLNB,FIRST=BSF

 ALL DONE
XBS
    59.    61.    63.    65.    67.    69.    71.
    73.    75.    77.    79.    81.    83.    85.
    87.    89.    91.    93.    95.    97.    99.
   101.   103.   105.   107.   109.   111.   113.
   115.   117.   119.   121.   123.   125.   127.
   129.   131.   133.   135.   137.   139.   141.
   143.   145.   147.   149.   151.   153.   155.
   157.   159.   165.   166.   167.   168.   169.
  1773.
STOP
```

The last number listed (1773) is the number of machine cycles elapsing between the two uses of IRTC in the code ABCSF.

Notice, next, the result of an apparently innocuous change to line 2.

```
rp2!=BSF!=DABCS

.nf!run
     17 LINES (  80S)
CFT I=ABCSF,ON=G,L=L=LSF,B=FSF
FT004 - CFT VERSION -      04/06/79 SCHEDULER
FT001 - COMPILE TIME =       0.0191 SECONDS

 ALL DONE
LDR I=(BSF,BABCS),ML=MSF,X=XBS,ORDER=CLNB,FIRST=BABCS

 ALL DONE
XBS
    59.    61.    63.    65.    67.    69.    71.
    73.    75.    77.    79.    81.    83.    85.
    87.    89.    91.    93.    95.    97.    99.
   101.   103.   105.   107.   109.   111.   113.
   115.   117.   119.   121.   123.   125.   127.
   129.   131.   133.   135.   137.   139.   141.
   143.   145.   147.   149.   151.   153.   155.
   157.   159.   165.   166.   167.   168.   169.
  1659.
STOP

 ALL DONE
```

Other Methods

One can use the 072 machine instruction directly to discover ultra-fine timing details related to hardware and special code loops. This detail is made available to the CRAY-1 programmer through use of the public file "CYCLES". See Section IV for more information.

III. PREDICTING TIMING

The rest of this paper will be used to demonstrate (and, I hope, teach you) a method for explicitly predicting timing. The method can help in avoiding unnecessary conflicts in assembly-language-coded subroutines or in loops which one expects to utilize considerable machine time and for which, therefore, one is justified in spending considerable human time to obtain top performance. Since the method outlined is almost completely mechanical, a program using these ideas has been written to generate timing charts such as those shown below. The program is called CYCLES. Its usage is described in Section IV of this report.

I will assume that the reader is familiar with the CRAY-1 Hardware Manual and CAL assembly language. In particular, the five pages of our Appendix A, taken from the CRAY-1 Hardware Manual, list much of the information needed for timing purposes. Examples will be either given in CAL or, on occasion, taken directly from the long listing of CFT or CIVIC.

General Remarks

In general, the time required to perform an algorithm depends on the specific instructions used to perform it and on the relationships among those instructions. A complete understanding of the relevant conditions affecting the execution of a particular instruction can be gained only by considering its relation to surrounding instructions. In particular, vector instructions require somewhat more analysis than scalars.

I find that recording at most five easily computed numbers per instruction will give the necessary information for determining conflicts and suggesting ways to avoid them. For a scalar (or register) instruction one needs to keep track of: (1) when it issues, and (2) when it completes. For a vector instruction one has to note: (1) its issue time, (2) its chain time, and the (different) times when it has finished using: (3) its input registers, (4) its functional unit, and (5) its output register.

In all cases, except for scalar memory-referencing instructions (and normally it is true then, also), once the issue cycle has been determined, all the other timing numbers for that instruction are computable. The rules for doing these computations are stated on page 25 of this report, and the exceptions are noted in appropriate examples.

Table 1 (adapted from Appendix D of the CRAY-1 Hardware Manual) lists the entire set of timing numbers (first column) needed for most purposes. These specify the number of 12.5 nanosecond machine cycles required by the CRAY-1 to deliver a result to the appropriate register. (0 means no result goes to a register.) Further detail is available in Chapter 4 of the Cray-1 Hardware Manual in conjunction with each specific instruction description.

Note. All instructions using the Memory Functional Unit are subject to possible additional delays due to memory bank conflicts with I/O.

Table 1. Instruction and Timing Summary

Cy-cles	CRAY-1	CAL mnemonics		Unit	Description
∞	000xxx	ERR		-	Error exit
50	**000ijk	ERR	exp	-	Error exit
0	*001000	NOP		-	No operation
1	**0010jk	CA,Aj	Ak	-	Set the channel (Aj) current address to (Ak) and begin the I/O sequence
1	**0011jk	CL,Aj	Ak	-	Set the channel (Aj) limit address to (Ak)
1	**0012jx	CI,Aj		-	Clear channel (Aj) interrupt flag
1	**0013jx	XA	Aj	-	Enter XA register with (Aj)
1	**0014jx	RT	Sj	-	Enter real-time clock register with (Sj)
1	**0014j4	PCI	Sj	-	Enter II with (Sj)
1	**0014j5	CCI		-	Clear clock interrupt
1	**0014j6	ECI		-	Enable Clock interrupt
1	**0014j7	DCI		-	Disable clock interrupt
1	0020xk	VL	Ak	-	Transmit (Ak) to VL register
1	*0020x0	VL	1	-	Transmit 1 to VL register
1	0021xx	EFI		-	Enable interrupt on flt pt error
1	0022xx	DFI		-	Disable interrupt on flt pt error
3	003xjx	VM	Sj	-	Transmit (Sj) to VM register
3	*003x0x	VM	0	-	Clear VM register
∞	004xxx	EX		-	Normal exit
50	**004ijk	EX		-	Normal exit
7(+)	005xjkx	J	Bjk	-	Jump to (Bjk)
5(+)	006ijkm	J	exp	-	Jump to exp
5(+)	007ijkm	R	exp	-	Return jump to exp; set B00 to P
5(+)	010ijkm	JAZ	exp	-	Branch to exp if (A0) = 0
5(+)	011ijkm	JAN	exp	-	Branch to exp if (A0).NE.0
5(+)	012ijkm	JAP	exp	-	Branch to exp if (A0) positive
5(+)	013ijkm	JAM	exp	-	Branch to exp if (A0) negative
5(+)	014ijkm	JSZ	exp	-	Branch to exp if (S0) = 0
5(+)	015ijkm	JSN	exp	-	Branch to exp if (S0).NE.0
5(+)	016ijkm	JSP	exp	-	Branch to exp if (S0) positive
5(+)	017ijkm	JSM	exp	-	Branch to exp if (S0) negative
1	020ijkm			-	Transmit exp = jkm to Ai
1	021ijkm	Ai	exp	-	Transmit exp = 1's complement of jkm to Ai
1	022ijk	Ai	exp	-	Transmit exp = jk to Ai
1	023ijx	Ai	Sj	-	Transmit (Sj) to Ai
1	024ijk	Ai	Bjk	-	Transmit (Bjk) to Ai

---

* Special CAL syntax form.
** Privileged to monitor mode.
x Indicates that the field is not used by the hardware; the assembler generates a zero in this position.
\+ These jump instructions take longer if branched-to address is not already in an instruction buffer. They then use the memory functional unit.

Cycles	CRAY-1	CAL mnemonics		Unit	Description
1	025ijk	Bjk	Ai	-	Transmit (Ai) to Bjk
4	026ix0	Ai	PSj	Pop/LZ	Population count of (Sj) to Ai
4	026ij1	Ai	QSj	Pop/LZ	Pop count parity of (Sj) to Ai
3	027ijx	Ai	ZSj	Pop/LZ	Leading zero count of (Sj) to Ai
2	030ijk	Ai	Aj+Ak	A Int Add	Integer sum of (Aj) and (Ak) to Ai
2	*030i0k	Ai	Ak	A Int Add	Transmit (Ak) to Ai
2	*030ij0	Ai	Aj+1	A Int Add	Integer sum of (Aj) and 1 to Ai
2	031ijk	Ai	Aj-Ak	A Int Add	Integer difference of (Aj) less (Ak) to Ai
2	*031i00	Ai	-1	A Int Add	Transmit -1 to Ai
2	*031i0k	Ai	-Ak	A Int Add	Transmit the negative of (Ak) to Ai
2	*031ij0	Ai	Aj-1	A Int Add	Integer difference of (Aj) less 1 to Aj
6	032ijk	Ai	Aj*Ak	A Int Mult	Integer product of (Aj) and (Ak) to Ai
4	*033i0x	Ai	CI	-	Channel number to Ai (j=0)
4	*033ij0	Ai	CA,Aj	-	Address of channel (Aj) to Ai (j.NE.0)
4	033ij1	Ai	CE,Aj	-	Error flag of channel (Aj) to Ai (j.NE.0)
14(+)	034ijk	Bjk,Ai	,A0	Memory	Read (Ai) words to B register jk from (A0)
14(+)	*034ijk	Bjk,Ai	0,A0	Memory	Read (Ai) words to B register jk from (A0)
6(+)	035ijk	,A0	Bjk,Ai	Memory	Store (Ai) words at B register jk to (A0)
6(+)	*035ijk	0,A0	Bjk,Ai	Memory	Store (Ai) words at Be register jk to (A0)
14(+)	036ijk	Tjk,Ai	,A0	Memory	Read (Ai) words to T register jk from (A0)
14(+)	*036ijk	Tjk,Ai	0,A0	Memory	Read (Ai) words to T register jk from (A0)
6(+)	037ijk	,A0	Tjk,Ai	Memory	Store (Ai) words at T register jk to (A0)
6(+)	*037ijk	0,A0	Tjk,Ai	Memory	Store (Ai) words at T register jk to (A0)
1	040ijkm	Si	exp	-	Transmit jkm to Si
1	041ijkm	Si	exp	-	Transmit exp = 1's complement of jkm to Si
1	042ijk	Si	<exp	S Logical	Form 1's mask exp = 64-jk bits in Si from the right
1	*042ijk	Si	#>exp	S Logical	Form 0's mask exp = jk bits in Si from the left
1	*042i00	Si	-1	S Logical	Enter -1 into Si

---

* Special CAL syntax form.
\+ The cycles needed = this number + (Ai). Also, no issues allowed till completion.
x Field not used.

Cycles	CRAY-1	CAL mnemonics	Unit	Description
1	*042i77	Si 1	S Logical	Enter 1 into Si
1	043ijk	Si >exp	S Logical	Form 1's mask exp = jk bits in Si from the left
1	*043ijk	Si #<exp	S Logical	Form 0's mask exp = 64-jk bits in Si from the right
1	*043i00	Si 0	S Logical	Clear Si
1	044ijk	Si Sj&Sk	S Logical	Logical product of (Sj) and (Sk) to Si
1	*044ij0	Si Sj&SB	S Logical	Sign bit of (Sj) to Si
1	*045ijk	Si #Sk&Sj	S Logical	Logical product of (Sj) and 1's complement of (Sk) to Si
1	*045ij0	Si #SB&Sj	S Logical	(Sj) with sign bit cleared to Si
1	046ijk	Si Sj\Sk	S Logical	Logical difference of (Sj) and (Sk) to Si
1	*046ij0	Si Sj\SB	S Logical	Toggle sign bit of Sj, then enter into Si
1	*046ij0	Si SB\Sj	S Logical	Toggle sign bit of Sj, then enter into Si (j.NE.0)
1	047ijk	Si #Sj\Sk	S Logical	Logical equivalence of (Sk) and (Sj) to Si
1	*047i0k	Si #Sk	S Logical	Transmit 1's complement of (Sk) to Si
1	*047ij0	Si #Sj\SB	S Logical	Logical equivalence of (Sj) and sign bit to Si
1	*047i00	Si #SB	S Logical	Enter 1's complement of sign bit into Si
1	050ijk	Si Sj!Si&Sk	S Logical	Logical product of (Si) and (Sk) complement ORed with logical product of (Sj) and (Sk) to Si
1	*050ij0	Si Sj!Si&SB	S Logical	Scalar merge of (Si) and sign bit of (Sj) to Si
1	051ijk	Si Sj!Sk	S Logical	Logical sum of (Sj) and (Sk) to Si
1	*051i0k	Si Sk	S Logical	Transmit (Sk) to Si
1	*051ij0	Si Sj!SB	S Logical	Logical sum of (Sj) and sign bit to Si
1	*051i00	Si SB	S Logical	Enter sign bit into Si
2	052ijk	S0 Si<exp	S Shift	Shift (Si) left exp = jk places to S0
2	053ijk	S0 Si>exp	S Shift	Shift (Si) right exp = 64-jk places to S0
2	054ijk	Si Si<exp	S Shift	Shift (Si) left exp = jk places
2	055ijk	Si Si>exp	S Shift	Shift (Si) right exp = 64-jk places
3	056ijk	Si Si,Sj<Ak	S Shift	Shift (Si and Sj) left (Ak) places to Si
3	*056ij0	Si Si,Sj<1	S Shift	Shift (Si and Sj) left one place to Si

---

* Special CAL syntax form.

Cycles	CRAY-1	CAL mnemonics		Unit	Description
3	*056i0k	Si	Si<Ak	S Shift	Shift (Si) left (Ak) places to Si
3	057ijk	Si	Sj,Si>Ak	S Shift	Shift (Sj and Si) right (Ak) places to Si
3	*057ij0	Si	Sj,Si>1	S Shift	Shift (Sj and Si) right one place o Si
3	*057i0k	Si	Si>Ak	S Shift	Shift (Si) right (Ak) places to Si
3	060ijk	Si	Sj+Sk	S Int Add	Integer sum of (Sj) and (Sk) to Si
3	061ijk	Si	Sj-Sk	S Int Add	Integer difference of (Sj) and (Sk) to Si
3	*061i0k	Si	-Sk	S Int Add	Transmit negative of (Sk) to Si
6	062ijk	Si	Sj+FSk	F.P. Add	Floating sum of (Sj) and (Sk) to Si
6	*062i0k	Si	+FSk	F.P. Add	Normalize (Sk) to Si
6	063ijk	si	Sj-FSk	F.P. Add	Floating difference of (Sj) and (Sk) to Si
6	*063i0k	Si	-FSK	F.P. Add	Transmit normalized negative of (Sk) to Si
7	064ijk	Si	Sj*FSk	F.P. Mult	Floating product of (Sj) and (Sk) to Si
7	065ijk	Si	Sj*HSk	F.P. Mult	Half precision rounded floating product of (Sj) and (Sk) to Si
7	066ijk	Si	Sj*RSk	F.P. Mult	Full precision rounded floating product of (Sj) and (Sk) to Si
7	067ijk	Si	Sj*ISk	F.P. Mult	2 - Floating product of (Sj) and (Sk) to Si
14	070ijx	Si	/HSj	F.P. Rcpl	Floating reciprocal approximation of (Sj) to Si
2	071i0k	Si	Ak	-	Transmit (Ak) to Si with no sign extension
2	071i1k	Si	+Ak	-	Transmit (Ak) to Si with sign extension
2	071i2k	Si	+FAk	-	Transmit (Ak) to Si as unnormalized floating point number
2	071i3x	Si	0.6	-	Transmit constant 0.75*2**48 to Si
2	071i4x	Si	0.4	-	Transmit constant 0.5 to Si
2	071i5x	Si	1.	-	Transmit constant 1.0 to Si
2	071i6x	Si	2.	-	Transmit constant 2.0 to Si
2	071i7x	Si	4.	-	Transmit constant 4.0 to Si
1	072ixx	Si	RT	-	Transmit (RTC) to Si
1	073ixx	Si	VM	-	Transmit (VM) to Si
1	074ijk	Si	Tjk	-	Transmit (Tjk) to Si
1	075ijk	Tjk	Si	-	Transmit (Si) to Tjk

---

* Special CAL syntax form.
x Field not used.

Cycles	CRAY-1	CAL mnemonics		Unit	Description
5	076ijk	Si	Vj,Ak	-	Transmit (Vj, element (Ak)) to Si
1	077ijk	Vi,Ak	Sj	-	Transmit (Sj) to Vi element (Ak)
1	*077i0k	Vi,Ak	0	-	Clear Vi element (Ak)
11	10hijkm	Ai	exp,Ah	Memory	Read from ((Ah) + exp) to Ai (A0=0)
11	*100ijkm	Ai	exp,0	Memory	Read from (exp) to Ai
11	*100ijkm	Ai	exp,	Memory	Read from (exp) to Ai
11	*10hi000	Ai	,Ah	Memory	Read from (Ah) to Ai
0	11hijkm	exp,Ah	Ai	Memory	Store (Ai) to (Ah) + exp (A0=0)
0	*110ijkm	exp,0	Ai	Memory	Store (Ai) to exp
0	*110ijkm	exp,	Ai	Memory	Store (Ai) to exp
0	*11hi000	,Ah	Ai	Memory	Store (Ai) to (Ah)
11	12hijkm	Si	exp,Ah	Memory	Read from ((Ah) + exp) to Si (A0=0)
11	*120ijkm	Si	exp,0	Memory	Read from exp to Si
11	*120ijkm	Si	exp,	Memory	Read from exp to Si
11	*12hi000	Si	,Ah	Memory	Read from (Ah) to Si
0	13hijkm	exp,Ah	Si	Memory	Store (Si) to (Ah) + exp (A0=0)
0	*130ijkm	exp,0	Si	Memory	Store (Si) to exp
0	*130ijkm	exp,	Si	Memory	Store (Si) to exp
0	*13hi000	,Ah	Si	Memory	Store (Si) to (Ah)
4	140ijk	Vi	Sj&Vk	V Logical	Logical products of (Sj) and (Vk) to Vi
4	141ijk	Vi	Vj&Vk	V Logical	Logical products of (Vj) and (Vk) to Vi
4	142ijk	Vi	Sj!Vk	V Logical	Logical sums of (Sj) and (Vk) to Vi
4	*142i0k	Vi	Vk	V Logical	Transmit (Vk) to Vi
4	143ijk	Vi	Vj!Vk	V Logical	Logical sums of (Vj) and (Vk) to Vi
4	144ijk	Vi	Sj\Vk	V Logical	differences of (Sj) and (Vk) to Vi
4	*145iii	Vi	0	V Logical	Clear Vi
4	145ijk	Vi	Vj\Vk	V Logical	Logical differences of (Vj) and (Vk) to Vi
4	146ijk	Vi	Sj!Vk&VM	V Logical	Transmit (Sj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
4	*146i0k	Vi	#VM&Vk	V Logical	Vector merge of (Vk) and 0 to Vi
4	147ijk	Vi	Vj!Vk&VM	V Logical	Transmit (Vj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
6	150ijk	Vi	Vj<Ak	V Shift	Shift (Vj) left (Ak) places to Vi
6	*150ij0	Vi	Vj<1	V Shift	Shift (Vj) left one place to Vi
6	151ijk	Vi	Vj>Ak	V Shift	Shift (Vj) right (Ak) places to Vi
6	*151ij0	Vi	Vj>1	V Shift	Shift (Vj) right one place to Vi
6	152ijk	Vi	Vj,Vj<Ak	V Shift	Double shift (Vj) left (Ak) places to Vi

---

* Special CAL syntax form.

Cycles	CRAY-1	CAL mnemonics	Unit	Description
6	*152ij0	Vi Vj,Vj<1	V Shift	Double shift (Vj) left one place to Vi
6	153ijk	Vi Vj,Vj>Ak	V Shift	Double shift (Vj) right (Ak) places to Vi
6	*153ij0	Vi Vj,Vj>1	V Shift	Double shift (Vi) right one place to Vi
5	154ijk	Vi Sj+Vk	V Int Add	Integer sums of (Sj) and (Vk) to Vi
5	155ijk	Vi Vj+Vk	V Int Add	Integer sums of (Vj) and (Vk) to Vi
5	156ijk	Vi Sj-Vk	V Int Add	Integer differences of (Sj) and (Vk) to Vi
5	*156i0k	Vi -Vk	V Int Add	Transmit negative of (Vk) to Vi
5	157ijk	Vi Vj-Vk	V Int Add	Integer differences of (Vj) and (Vk) to Vi
9	160ijk	Vi Sj*FVk	F.P. Mult	Floating products of (Sj) and (Vk) to Vi
9	161ijk	Vi Vj*FVk	F.P. Mult	Floating products of (Vj) and (Vk) to Vi
9	162ijk	Vi Sj*HVk	F.P. Mult	Half precision rounded floating products of (Sj) and (Vk) to Vi
9	163ijk	Vi Vj*HVk	F.P. Mult	Half precision rounded floating products of (Vj) and (Vk) to Vi
9	164ijk	Vi Sj*RVk	F.P. Mult	Rounded floating products of (Sj) and (Vk) to Vi
9	165ijk	Vi Vj*RVk	F.P. Mult	Rounded floating products of (Vj) and (Vk) to Vi
9	166ijk	Vi Sj*IVk	F.P. Mult	2 - floating products of (Sj) and (Vk) to Vi
9	167ijk	Vi Vj*IVk	F.P. Mult	2 - floating products of (Vj) and (Vk) to Vi
8	170ijk	Vi Sj+FVk	F.P. Add	Floating sums of (Sj) and (Vk) to Vi
8	*170i0k	Vi +FVk	F.P. Add	Normalize (Vk) to Vi
8	171ijk	Vi Vj+FVk	F.P. Add	Floating sums of (Vj) and (Vk) to Vi
8	172ijk	Vi Sj-FVk	F.P. Add	Floating differences of (Sj) and (Vk) to Vi
8	*172i0k	Vi -FVk	F.P. Add	Transmit normalized negatives of (Vk) to Vi
8	173ijk	Vi Vj-FVk	F.P. Add	Floating differences of (Vj) and (Vk) to Vi
16	174ij0	Vi /HVj	F.P. Rcpl	Floating reciprocal approximations of (Vj) to Vi
8	174ij1	Vi PVj	F.P. Rcpl	Population counts of (Vj) to Vi
8	174ij2	Vi QVj	F.P. Rcpl	Pop count parity of (Vj) to Vi

---

* Special CAL syntax form.

Cycles	CRAY-1	CAL mnemonics	Unit	Description
6	175xj0	VM Vj,Z	V Logical	VM=1 where (Vj) = 0
6	175xi1	VM Vi,N	V Logical	VM=1 where (Vj).NE.0
6	175xj2	VM Vj,P	V Logical	VM=1 where (Vj) positive
6	175xj3	VM Vj,M	V Logical	VM=1 where (Vj) negative
9	176ixk	Vi ,A0,Ak	Memory	Read (VL) words to Vi from (A0) incremented by (Ak)
9	*176ix0	Vi ,A0,1	Memory	Read (VL) words to Vi from (A0) incremented by 1
0	177xjk	,A0,Ak Vj	Memory	Store (VL) words from Vj to (A0) incremented by (Ak)
0	*177xj0	,A0,1 Vj	Memory	Store (VL) words from Vj to (A0) incremented by 1

---

* Special CAL syntax form.
x Field not used.

The Basic Details

In general we have the following scenario: in order to perform some alteration of the contents of one or more of the machine's registers or memory, an instruction must: first, wait to be brought into one of the instruction buffers; second, wait until prior instructions have started; third, wait till its operands are available; and fourth, wait until all shared components (such as paths along which information may flow, registers that may be needed, and functional units that may be employed) will be available during the cycle(s) required. The CRAY-1 hardware maintains reservation tables, updated each cycle, for each register and all other shared components. It releases or issues an instruction only when it can be completed without interference from other previously issued instructions.

Generally, timing analysis begins when the first instruction of interest issues, but it is naive not to consider its placement in an instruction buffer and the route by which it reached issuable condition. For many algorithms, speed changes on the order of 10% occur depending on their placement relative to the start of an instruction buffer. Details about the instruction fetch mechanism are found in Appendix C.

All of the information used to decide about the issue of an instruction is contained in its 16 bits or, in the case of a 32-bit instruction, in its upper 16 bits. Normally the decision to issue can be made in one cycle. When an instruction issues, the components it will use are reserved in the appropriate table for the appropriate time period.

One type of 32-bit instruction, which makes a scalar memory reference, is allowed to issue when all of the components it will need are available except possibly the appropriate memory bank. If the bank is available at the proper time, all proceeds normally. If not, completion of the instruction is delayed and the next instruction requesting memory is not allowed to issue until the previous one has obtained the proper memory access. Instructions not requiring memory, however, may proceed normally.

Until a specific instruction issues, the machine cannot look beyond it to determine that something further down in the instruction sequence could be done. It is the task of the programmer and compiler to so order the computation that unnecessary delays are avoided. When you program in assembly language, it is important (and not difficult) to maintain an understanding of the resources of the machine called into play by each instruction and of the cycles in which they are used, in order to approach optimum utilization of the hardware.

During the issue cycle, paths are opened so that information can flow from registers to functional units; during the completion cycle, paths are

required for information to flow from functional units to registers. Only one path is available to service all results being returned to any of the eight S-registers. There is also one path for the A-registers. Possible conflicts over the use of these paths are resolved before an instruction is allowed to issue. A separate path into and out of each vector register is provided. Moreover, information arriving at any register in a given cycle may also be redirected by a subsequent instruction, in that same cycle, to serve as input for another operation. That is, a subsequent instruction may issue on the same cycle in which its operands first become available. This redirection of information arriving at a vector register is called chaining, and it may begin only during the particular cycle when the first element of the result is returned from a functional unit. If two different functional units return their first results in the same cycle, a third instruction may chain from both of them.

An exception to this "same cycle rule" occurs for conditional branch instructions, which require that their operand register becomes available somewhat before issue.

Two Short Examples

Let us consider what the hardware must take into account to decide when to issue a couple of typical instructions.

First, a scalar floating point add: 62312, S3 S1+FS2.

When the instruction sequence reaches such an instruction, the hardware checks its reservation tables to see that none of the following conditions are true: (1) the floating point add functional unit is busy (i.e., reserved) in this cycle, (2) register S3 is busy, (3) register S1 is busy, (4) register S2 is busy, (5) a reservation exists for the S-register input path 6 cycles hence. If any of these conditions are true, the instruction does not issue. In the next cycle (the machine having updated all its tables), the same conditions are tested. Eventually, all the needed components will be free and the instruction will issue. When it does, the tables will have: (1) a busy condition placed on S3 for 6 cycles (i.e., cycles 0,1,2,3,4, and 5) and (2) a reservation placed on the S-register input path 6 cycles hence (cycle 6). (No reservation is put on a functional unit by a scalar instruction.) In the next cycle, the next instruction will be considered for issue, and the components it needs will be checked for availability.

Now consider a vector instruction: 171312, V3 V1+FV2.

When this floating point vector add is reached, the hardware checks its reservation tables for the following conditions: (1) floating point adder reserved, (2) vector register V3 busy, (3) V1 busy, and (4) V2 busy. It does not need to check for path reservations since each V-register has its own path. When none of these conditions are true, the instruction issues. When

it does, (1) the tables have a busy condition placed on V1 and V2 for, max((VL),5) cycles, where (VL) is the current value of the vector length register (thus for short vectors a minimum reservation of 5 cycles occurs), (2) a busy is placed on the floating point adder for (VL)+4 cycles, (3) a busy is placed on V3 for cycles 1 through 7 and cycles 9 through 7+max((VL),5). Cycle 8 is the "chain" cycle.

The Timing Chart

We can keep track of important cycles by listing them in a timing chart. Then, when we want to consider whether a particular instruction can issue, we have the information at hand. In practice, it is easier to list the cycles when a component will next become ready for use than to record those in which it is busy.

In such a chart, I and C refer to issue cycle and completion cycle for scalars, respectively, while I,C,Ø,F, and R refer to issue cycle, chain cycle, operand register ready cycle, functional unit available cycle, and result register ready cycle for vectors.

Thus we have:

			I	C	Ø	F	R
62312	S3	S1+FS2	0	6			

while, supposing the following instruction comes in sequence with the above and that (VL) = 64:

171312	V3	V1+FV2	1	9	65	69	73.

The numbers recorded in the various columns represent the cycles in which certain important changes will occur as a result of the issue of the instruction in question. (Since for scalar instructions, the last three columns are not particularly informative, one may omit them.) Different types of instructions tie up different machine resources for differing numbers of cycles, as indicated in Table 1. (See also Appendices A and D of the CRAY-1 Hardware Manual.) In the examples that follow, we will demonstrate the practical use of these timing numbers. In general, the entry in the C column is the I number plus the appropriate instruction execution-complete time from the first column of Table 1.

Preliminary Considerations

Consider the first add mentioned above: 62312, with I = 0 and C = 6. The 6 has two meanings. First, it is the cycle on which the result will be returned to S3 via the S-register output path. This means that this number cannot appear as the C cycle for any other (later issued) instruction whose result is destined for any S-register. For example, if the next instruction

were 76567, transmit a V-register element to S5, which takes 5 cycles, then the machine must delay its issue. If you are recording the I and C numbers for a series of instructions, you should notice when you record two identical numbers in the C column. If the second is a result for the same set of registers as the first, it will be delayed, and you must adjust the issue cycle accordingly.

Secondly, the 6 has another meaning. Cycle 6 is also the cycle on which the register becomes available for use (either as an operand or a result) by another instruction. For example, in coding a set of instructions, one might attempt to reuse an S-register before it has completed a previous operation. Thus, one might do a reciprocal into S6 and then read the time clock into S6. The timing is then:

```
          I     C
70610     0     14
72600    14     15
```

since the result of the clock read is not allowed to use S6 until the reciprocal is through with it. This assures that the result of the reciprocal will be overwritten by the later instruction.

It is perhaps more common that a later instruction which would use the result of the reciprocal as an operand, would have to wait for it. Thus:

```
          I     C
70610     0     14
67561    14     21
```

would be the timing for these two instructions.

For vector instructions, the relations among the numbers I, C, Ø, F, and R, are found as follows: When the issue time I becomes known, then C will be equal to I + the chain time for this instruction (the chain time being the functional unit time + 2), Ø will equal I +(VL), F = I+4+(VL) (thus F will normally be Ø+4) (here, however, one exception exists, for vector store F = I+5+(VL)), and finally R = C + (VL). For short vectors, where (VL) ≤ 4, C and F are as before, while Ø = I+5 and R = C+5.

Thus if (VL) = 2, we have:

```
            I   C   Ø   F    R
171312      1   9   6   7   14 .
```

All five vector timing numbers depend only on the chain (C) cycle (from Table 1), (VL), and issue (I).

Two Basic Examples and Comments

In the two examples below, taken from (more or less) real programs, nearly all of the main ideas surrounding accurate timing of code are mentioned. Examine the instruction sequence and refer to the notes for an explanation of the timing numbers listed.

Example 1

First, we consider the earlier example, ABC:

```
 1 *      CFT I=ABCSF,ON=G,L=LSF,B=BSF
 2 *       LDR I=(BSF,BABCS),ML=MSF,X=XBS,ORDER=CLNB,FIRST=BABCS
 3 *     XBS
 4         COMMON /ABCOMMON/ A(56),OUTRANGE,B(56),C(56)
 5         CALL LINK('UNIT59=TERMINAL//')
 6         Y = X*X*X*X*X*X*Y*X
 7         DO 1 I = 1,169
 8    1    A(I) = I
 9         OUTRANGE = 6000040000000000000000B
10         M = IRTC(0)
11         CALL ABCS
12         N = IRTC(0)
13         X = N-M
14         WRITE(59,59) C,X
15   59    FORMAT(7F6.0)
16         STOP
17         END
```

ABC consists of a FORTRAN part, ABCSF (MAIN.), where the RTC is read, and a CAL part ABCS, where adds are done. We note that we are timing the case where the assembly portion is loaded first.

Listed below is the set of six assembly instructions generated by CFT for the portion of the code where the RTC read occurs (extracted from the long listing). The address listed is after the load. Recall that I and C refer to the machine cycle on which instruction issue and completion, respectively, occur (see Table 1). (The small letters refer to notes following.)

Address	Machine code (octal)		Mnemonics (decimal)		I	C	Comment
251a	072300		S3	RT	0e	1f	Read RTC
251b	130300	000225	M,0	S3	1g	-h	Save RTC
251d	022700		A7	0	3i	4j	Arg count
252a	007000	001000	R	ABCS	4	19k	Call subroutine
252c	120100	000225	S1	M,0	1657m	1668n	Get saved RTC
253a	072700		S7	RT	1659	1660	Read new RTC

Notes: (a,b,c,d at the left refer to the parcel address in the word where the instruction is located.)

e. Assume all resources of the machine are available, initially.
f. A "72" instruction requires one cycle to complete after issue (see Table 1). If any previously issued instruction had needed to put a result into any S-register during cycle 1, the issue of this instruction would have to be delayed by the machine.
g. The instruction following a 16-bit instruction may issue on the next cycle (if there is no conflict, as is the case here), S3 being now available.
h. A store instruction uses an S or A register only during the issue cycle. The result actually reaches memory several cycles later, but for purposes of subsequent fetch instructions, vector loads, or memory busy conditions, the memory is essentially free after four cycles, while the register itself remains free.
i. The instruction following a 32-bit instruction may not issue until after a delay of one cycle (to bypass the lower 16 bits).
j. A "22" instruction requires one cycle to complete after issue. If a previously issued instruction needed to put a result into any A-register during cycle 4, this issue would be delayed. (But an S-reg result could complete then without delaying this.)
k. This instruction, which would normally complete at cycle 18, is delayed for one cycle by memory busy from the previous store, since a memory-busy condition is not allowed when starting the fetch of the next 16-word buffer-load of instructions. If this "007" instruction addressed an instruction from code already in a buffer, it would complete at cycle 9. In the case of a jump instruction, completion means that the jumped-to instruction may issue.
m. This fetch instruction cannot issue until the called subroutine returns to it. See the analysis of ABCS below.
n. When it does issue it will require 11 cycles for the contents of memory to reach the S-register. The memory bank will be free after only four cycles.

Now consider the CAL portion of our example, called by the FORTRAN portion above.

```
                                *       CAL  I=ABCS,B=BABCS,L=LSC
                                          IDENT             ABCS
                                          COMMON   ABCOMMON
                          71    A         BSS               57
                          70    B         BSS               56
                          70    C         BSS               56
                                          BLOCK             ABCS
                                          ENTRY             ABCS
022100                            ABCS     A1                0
022263                                     A2                51
1211 000000000C                    LOOP    S1                A,A1
1212 000000071C                            S2                B,A1
062312                                     S3                S1+FS2
1313 00000161C                             C,A1              S3
030110                                     A1                A1+1
031012                                     A0                A1-A2
011 00000000c+                             JAN               LOOP
005000                                     J                 B00
                                           END
```

Since the instructions here form a loop to be performed 51 times, we must consider them more than once. The instructions for pass 1 are:

Address	Machine code (octal)		Mnemonics (decimal)		I	C
200a	022100		A1	0	19k	20
200b	022263		A2	51	20	21
200c	121100	025511	S1	A,A1	21	32
201a	121200	025602	S2	B,A1	23	34
201c	062312		S3	S1+FS2	34p	40q
201d	131300	025662	C,A1	S3	40	-r
202b	030110		A1	A1+1	42	44s
202c	031012		A0	A1-A2	44	46
202d	011000	001002	JAN	LOOP	48t	53u
(203b	005000)		(J	B00)	(50	57)v

Notes for pass 1:

k. See previous note k.

p. The issue of the add instruction is delayed until both operands (S1 and S2) have arrived from memory. The completion cycle of the S2 fetch is the start cycle of the add.

q. A floating point add requires six cycles to complete (from Table 1).

r. Normally, we don't need to consider memory. S3 is available to start the store at cycle 40, and remains available for other use in the next cycle.

s. An address add requires two cycles. (So does an A to A move, which is really an add of 0.)

t. A conditional jump instruction does not issue until two cycles after the needed operand becomes available. (A0 is returned at 46; 47 is skipped; 48 is issue.) Other instructions, even one using A0 (but not putting a result into A0) could issue at 47, and the jump would still go at 48.

u. This in-stack branch (to 200c) requires five cycles.

v. The numbers here refer to the cycles on which this instruction would have issued and completed, if the program did not branch back.

The instructions and timing for passes 2 and 51 are as follows

Address	Machine code (octal)		Mnemonics (decimal)	I	C
Pass 2					
200c	121100	025511	S1 A,A1	53u	64
	. . .		(add 32 to Pass 1 numbers)		
202d	011000	001002	JAN L00P	70	75
(203b			J B00	72	79)v
	. . .				
Pass 51					
			(add 1600 to Pass 1. number)		
202d	011000	001002	JAN L00P	1648	1653
203b	005000		J B00	1650w	1657x
252c	120100	000225	S1 M,	1657	1668
253a	072700		S7 RT	1659	1660
253b	120200	000225	S2 M,	1660	1672y
253d	120300	000225	S3 M,	1663y	1676

Notes for Passes 2 through 51:

u. The in-stack branch completes and this instruction issues during cycle 53.

v. Once again, these are the "if it didn't" times.

w. This time it doesn't.

x. The return jump requires only seven cycles to complete because the code that called this routine is still in a buffer.

y. Consecutive scalar loads (or stores) may issue as few as 2 cycles apart and, if they do not address the same memory bank, finish in 11 additional cycles. If the second does address the same bank, it will require one or two extra cycles to finish, and a third consecutive scalar load (or store) will be delayed from issue until memory is free (at most four cycles later).

In general, a scalar load or store that encounters a memory conflict (which could come from I/O), issues as usual. This allows subsequent nonmemory instructions to proceed normally, while delaying memory instructions until the conflict is resolved. On the other hand, vector loads or stores (and instruction-buffer loading) wait until memory is entirely free before issuing (or starting). Such delays usually last no more than two cycles.

The cycles listed above are the actual machine cycles on which the events happen for the sequence of instructions given. It should be clear, however, that we could predict these numbers from the timing information in Table 1, together with a minimal understanding of the material from Appendix A (with the exception, perhaps, of the memory conflict details). One simply proceeds line by line, recording the five columns of numbers, left to right.

Thus, given the task of writing an efficient scalar loop to compute C =A+B, we can try a few alternate ways to do it, timing each one as we go, until we have identified the one with the lowest last-issue cycle.

For example, changing the three lines

```
     C,A1    S3
     A1      A1+1
     A0      A1-A2
to
     A1      A1+1
     A0      A1-A2
     C-1,A1  S3
```

would cut six cycles from the loop time and thus result in nearly a 20% saving in the measured execution time, (26 rather than 32 cycles per loop).

While it is actually possible to accomplish this loop by a scalar method in 14 cycles per pass, the parallel, nonrecursive nature of the loop allows a much greater saving by using vector instructions. So, let us now consider

code ABCV, and list its timing details. For an alternate view, we use CIVIC for this compilation.

Example 2

```
                *    CIVIC ABCVF CVF BVF LVF P24 L
                *    LDR I=(BABCV,BVF),ML=MVF,X=XVF
                * XVF
                      COMMON /ABCOMMON/ A(56),OUTRANGE,B(56),C(56)
000000A               CALL LINK('UNIT59=TERMINAL//')
000001D               DO 1 I = 1,169
000002C          1    A(I) = I
000010D               OUTRANGE = 6000040000000000000000B
000011D               M = Q8RTC(0)
000012C               CALL ABCV
000013B               N = Q8RTC(0)
000014B               X = N-M
000014A               WRITE(59,59) C,X
                 59   FORMAT(7F6.0)
000037C               STOP
                      END
                            *   CAL I=ABCV,E=X00,B=BABCV,L=LVC
                                      IDENT     ABCV
                                      COMMON    ABCOMMON
                             71   A           BSS         57
                             70   B           BSS         56
                             70   C           BSS         56
                                      BLOCK     ABCV
                                      ENTRY     ABCV
                   022363             ABCV     A3          51
                   0200 000000000C               A0          A
                   002003                        VL          A3
                   176100                        V1          ,A0,1
                   0200 000000071C               A0          B
                   176200                        V2          ,A0,1
                   171312                        V3          V1+FV2
                   0200 000000161C               A0          C
                   177030                        ,A0,1       V3
                   005000                        J           B00
                                                 END
```

Again, we consider the code from one read RTC to the next. Note that since this particular set of adds is not more than 64 in length, it can be done without looping instructions.

We will now record the full five columns of numbers. The I, C, O, F, and R refer to issue cycle, chain cycle for vector instructions (or completion cycle for scalers), operand register(s) free cycle, functional unit free cycle, and result register free cycle, respectively.

Address	Machine code (octal)		Mnemonics (decimal)		I	C	Ø	F	R
5013d	072300		S3	RT	0	1			
5014a	130300	005053	M,	S3	1	5			
5014c	022700		A7	0	3	4			
5014d	007000	024000	R	ABCV	4	9e			
5000a	022363		A3	51	9	10			
5000b	020000	000200	A0	A	10	11			
5000d	002003		VL	A3	12	13			
5001a	176100		V1	,A0,1	13f	22g	-h	68i	73j
5001b	020000	000271	A0	B	14k	15			
5001d	176200		V2	,A0,1	68l	77	-	123	128
5002a	171312		V3	V1*FV2	77m	85n	128o	132n	136n
5002b	020000	000361	A0	C	78	79			
5002d	177030		,A0,1	V3	136p	-q	187r	192s	-
5003a	005000		J	B00	137	144			
5015b	072100		S1	RT	144t	145			
5015c	130100	005054	N,	S1	192u	-			

Notes

(a, b, c, and d are parcel addresses, after the load, as before.)

e. For this compilation the destination of the return jump is already loaded into a buffer, so the branch instruction executes in only five cycles.

f. To begin execution, this vector instruction needs A0 and VL to be ready, V1 to be free, and memory to be free. Since they are, it issues.

g. The first result will be arriving from memory nine cycles after the issue cycle. This cycle (cycle 22) is the chain cycle for this memory load. (More on chaining in note m.)

h. When this instruction issues (cycle 13) it transmits as operands the contents of the VL register, the special value 1, and register A0 to the memory functional unit. (Some vector memory loads use a second A-register for the increment.) All these scalar transmissions occur during the issue cycle and are held by the functional unit thereafter. [When A0 and S0 are used as special values their reservation is not checked, and so they do not delay issue. Here, however, A0 is also used to hold an address, and if it had not been free when needed, the issue would be delayed.] For a vector load instruction, no vector register is used as input, so no entry is made in column Ø.

i. For this instruction, the functional unit involved is memory. As with scalar memory references, a memory bank will be busy for four cycles with

each word read. If the vector load moves through at least three other banks before returning to a previous one (as is the case here), no conflicts will arise, and a new word will be read each cycle. The first word is requested at cycle 13 and the 51st at cycle 63. The memory will be busy for 4 more cycles, through cycle 67, and free for another memory reference in the next cycle. We record 68 = 13+51+4 under the functional unit free column. Notice that memory is free five cycles before register V1 is ready.

j. When this instruction issues (cycle 13), it puts a hold, or reserve, on register V1 in order to keep it available for the words coming in from memory. The reserve will be lifted after the last word arrives. Since the (VL) is 51, the last (51st) word will arrive in cycle 72. (The first arrives in cycle 22.) In the next cycle the V1 register may be used for another purpose; therefore we record 73 = 22+51 under the result register free column. The CRAY hardware has one element pointer for each V-register, and it is used to select one of the 64 positions in the V-register. The pointer for register V1 is automatically stepped from 1 through 51 during cycles 22 through 72.

k. Since the previous vector instruction read out A0 and (VL), saving them in the functional unit at the start of the vector load, subsequent instructions may modify them immediately without affecting the previous instruction.

l. Here a major delay is encountered. This instruction also transmits words from memory to a V-register. The register is available but the memory is busy, so issue is delayed till it is free (in cycle 68).

m. This instruction chains. At cycle 69, it is first considered for issue. However, before it can begin executing, this vector add needs to have the vector length register, register V1, register V2, the floating point add functional unit, and register V3 free. V1, as noted, becomes free at cycle 73; V2 will not be free until 128; but the first element will arrive at cycle 77 and during that one cycle, it can be redirected, or chained, to serve as input to the add unit as well as being put into V3. The conditions for chaining are thus satisfied during cycle 77, and so the instruction issues.

n. The first result exits from the floating point adder eight cycles after the first operands were sent over. For this instruction, then, its chain cycle is 85 = 77+8. Similarly its result register (V3) free cycle is 136 = 85+51, and its functional unit free cycle is 132 = 77+51+4. The four extra cycles here are equivalent to the four extra cycles needed for memory free by the memory functional unit. All functional units remain reserved for four extra cycles after the last element arrives during vector instructions. This means that a subsequent scalar (or vector) floating point add cannot issue until cycle 132, since it shares this unit.

o. Since this instruction requires that vector register operands be sent to

the adder for the next 51 cycles, a reserve is placed on registers V1 and V2 until cycle 128, at which time they will both be free and able to be used by a subsequent operation.

p. This vector store does not chain from the add. In the first place, at cycle 85, the chain cycle for V3, the memory is busy completing the load of V2. In the second place, store instructions are barred by the hardware from chaining even if the memory functional unit is free. The store doesn't begin at cycle 123 (when the memory becomes free) either. It can't issue at 123 because the element pointer for V3 is not pointing to V3's first element, which the store needs, but rather at element 39, which is being returned by the floating-point adder. It finally issues when register V3 is not otherwise busy and can have its element pointer reset, namely cycle 136, the result register free cycle for the earlier add.

q. A store doesn't chain to anything, either.

r. Register V3 will be free after the store at cycle 187 = 136+51.

s. Finally, the memory functional unit will become free from the store five cycles after the operand register, V3, is free. All other instructions free their functional units four cycles after their operand registers but store requires one extra cycle.

t. Since the return from subroutine did not require memory, as the address is already in a buffer, the next instruction, which for CIVIC is the read of the RTC, gets issued well before the vector store completes.

u. Finally, we note that the final store of the RTC value to memory is delayed by the memory busy condition from the vector store, and issues when the memory functional unit ready cycle occurs.

Conclusions

It should be clear from the timing chart above that the CRAY-1 is not really very busy during this vector add routine. For example at cycle 78, its busiest cycle, V-registers 0,4,5,6, and 7 are free along with the shift, fixed add, multiply, reciprocal, and logical functional units. Moreover, the next 55 cycles (as well as most of the previous 60) could be used to issue independent instructions for a related calculation, if one needed to be done. (In fact, we can actually decrease the time for ABCV by four cycles by using some of the idle resources.)

Frequently, parallel use of available resources can be made, especially in the case of vector loops. Three examples of actual code are presented in Section V to show this: ZVSEEK, QVDIV0, and QVSQRTH.

IV. THE COMPUTER CODE CYCLES

CYCLES is a public file on the CRAY-1 computers at LLNL. It was written by Rollin Harding. A Fortran version of it has been made available to Cray Research Incorporated and is being modified for use under their system.

CYCLES is not a simulator and does not have knowledge of the values in all machine registers. It does, however, try to keep track of the values in the VL and A registers. Options allow these register values to be specified for CYCLES' use.

The rest of this section is taken from the documentation for CYCLES. A full writeup, CYCLEWUP, can be extracted from the CYCLES public file using LIB.

Cycles Writeup

CYCLES was designed for detailed analysis of instruction scheduling in compiled or assembled CRAY codes. The timing analysis is presented in the spirit of Harry Nelson's report, UCID-30179, Rev. 1, "Timing Codes on the CRAY-1". Harry supplied additional timing details and tested the code extensively during the debugging period.

Input to CYCLES is any HSP file from CAL, CIVIC, CFT, or DDT which contains the machine code listing. CYCLES accepts single or double column listings from CIVIC (M or L option) and the four instructions per line format from CFT (on=g). Sequences of octal parcels may be entered from TTY or by specifying octal word limits in a controllee or other binary file. In TTY or binary modes CYCLES adds the equivalent CRAY assembly language instructions to the output, i.e. does a CRAY UNDO. CYCLES will also accept the history file produced by DDT in the MNE output format mode. This form has the advantage of using correct symbols for variables in the program being undone.

Output consists of a copy of the input file with up to seven columns of timing information added for each machine instruction line. (This overwrites the comment field in CAL listings.) The NOCOPY. option will suppress most non-instruction lines from being output. The seven timing columns are:

W	number of cycles this instruction waited to issue
D	octal codes identifying any delays
I	issue cycle for the current instruction
C	vector chain cycle or scalar completion cycle
O	vector operand register ready time
F	vector functional unit ready time
R	vector result ready time

The I column is always given; others are suppressed if null or irrelevant for the current instruction. Alternate definitions for columns C, O, F, and R for jump instructions are given below.

CYCLES is very fast and is easily run as a controllee under TRIXGL. Output can be viewed without line wraparound by using TUBE command S or TRIXGL command TV,1 for small characters. Effects of altering instruction sequences can be checked easily by rearranging lines in CYCLES' infile and rerunning it without reassembling your code. One may also rearrange lines in CYCLES' outfile and then use that as the infile. CYCLES CIVIC output is compatible with single column CIVIC output. CYCLES' CFT, CAL, and binary output are compatible with CAL output. CYCLES' DDT output is compatible with DDT output.

Abilities and limitations

CYCLES is aware of most of the fine points of CRAY instruction scheduling:

- chaining requirements
- recursive vector operations
- no waits for special A0 and S0 operands
- memory functional unit requirements
- vector memory conflicts due to 8*n increments
- A and S register trunk conflicts.
- extra delay after A0 or S0 ready for conditional jumps.
- scalar memory bank conflicts (with limitations)
- instruction buffer fetches, conflicts, and delays.
- other special cases

CYCLES has to make assumptions about loader dependent conditions such as instruction buffer delays and scalar memory bank conflicts. Bank conflicts may not be detected if memory addresses are indefinite. Addresses are indefinite if they involve undefined A register values or unspecified relocation flags. Options are provided to specify that the current code block (local relocation) is loaded on a 20b-word buffer boundary or that all external blocks (subroutines or commons) are loaded on 20b-word boundaries. The relevant option names are +., x., and +x. to set relocation flags, and MBOFF. to turn off bank conflict checking. IBOFF. turns off instruction buffer checking.

VL and A registers

Many instruction timings depend on values of the vector length register and A registers. CYCLES attempts to keep VL and A regs current as instructions are processed that set those registers. A registers set from memory or from S registers are considered indefinite. Results of A register calculations involving indefinites are also indefinite. VL will be set to 64 if it is set from an indefinite A register. Register changes are reported in

the output. Automatic register setting can be disabled by the NOVLA. execute line option.

You may explicitly reset values for VL, A, or NI (next issue) by inserting control lines into CYCLES' input file or as comments in a CAL source file. In column 1 of the input file use Ln to set VL to n (decimal), use Cn to reset counters and force the next issue to cycle n (decimal), and use An,m to set register An to m (decimal). CAL comments *Ln, *Cn, and *An,m would have the same effects.

Jump instructions

For conditional jumps, CYCLES assumes drop through timing. Normally, the cycle counter is reset to zero after each unconditional jump. However, if the following instruction is recognized (by its address) as the target instruction, then timing continues without reset. This can be accomplished by control cards (CYCLE OFF/IN/OUT or REPEATn described below) or by rearranging the input file.

For a jump instruction certain columns are redefined:

C	Earliest issue for the jump target if the jump is taken
O	Target instruction buffer code (see I-buff section)
F	Target issue time for an in-buffer jump
R	Target issue time for an out-of-buffer jump

An out-of-buffer jump can be significantly delayed if memory is busy, for instance, completing a vector store.

You can control the output for a jump to a later instruction by inserting a control line CYCLE OFF immediately after the jump and a CYCLE IN or CYCLE OUT line immediately before the target instruction. CYCLES will stop timing after the OFF and will resume by issuing the target instruction at the proper IN buffer or OUT of buffer issue time. Comments, *CYCLE OFF, etc., can be used in a CAL source as well.

A REPEATn line can be used for continuous timing over a jump to an earlier instruction. The REPEAT line is inserted immediately before the target instruction. From then on, each jump instruction is checked to see if its target has an active repeat line. If it does, the count n is decremented, and timing continues at the target line using the in buffer time plus any appropriate delays for registers or functional units. Up to ten repeat lines may be active at any time. Repeats may be nested.

Instruction buffer (I-buff) delays

The CRAY has 4 instruction buffers. They are loaded in rotation. Each holds 20b words (64 parcels) of instructions. I-buff delays occur each time execution shifts from one buffer to another due to a jump instruction or

simply when crossing from one 20b block to the next. Additional delays result when memory operations conflict with instruction fetches or when a two-parcel instruction straddles a buffer boundary. For I-buff checking CYCLES assumes that relative word zero is loaded on a 20b-word boundary.

I-buff delays are indicated in the usual way, using delay code 200b, but additional information is also given:

- The first instruction from a buffer is marked (between the W and D columns) by a letter a,b,c, or d for buffer 0,1,2, or 3. Upper case means the instructions were fetched from memory; lower case means the buffer was already loaded.

- For jump instructions the target instruction buffer is given under the O column. Again, upper case is out-of-buffer; lower case is in-buffer. A jump to an external (x reloc) address is always considered out-of-buffer. An unconditional jump out-of-buffer clears one instruction buffer unless the NOICLR. option is used. A Bn line can be used to clear n additional instruction buffers.

- Delay code 10000b shows that an instruction fetch was delayed because memory was busy. Because of look-ahead, this does not cause an immediate delay of issue, but it does signal a possible delay for a subsequent issue (usually the target instruction of an out-of-buffer jump appearing in column F).

- Delay code 20000b indicates that the parcel address for the current instruction was not in a current I-buff or one that had been fetched. No delay is assessed.

- Delay code 40000b indicates the possibility of a delay that this version of CYCLES couldn't determine. The marked instruction is parcel 17c of the current instruction buffer. If the next instruction (17d) happens to be a two-parcel instruction (this is what the timing subroutine didn't know) then 17c would be delayed until one cycle before the issue time indicated on the next line for 17d. This delay of parcel 17c could cause further delays not shown for 17d, 20b, or later instructions. Correct timing can be produced in the current version by inserting an "In" control card before 17c, where n (decimal) is the correct issue time for 17c.

Availability of CYCLES

The latest version of CYCLES is maintained in CRAY public file CYCLES. The HELP packages are reproduced below. The output file is named Hinfile and is left on disk. An existing file will be overwritten. If the file overflows, sequence numbers will be added: 00, etc.

This writeup is available as CYCLEWUP in public file CYCLES. It will be revised as suggestions are made or changes made to CYCLES. The revision date is given on line 1.

```
    Please send suggestions for enhancements to CYCLES or listings of any
bugs you encounter to Rollin Harding in A-Division (L-16).

CYCLES HELP:

     execute lines:
   cycles hspfile type <nocopy. novla.  ...  noiclr. &> / t v
   cycles tty / t v
   cycles binfile fwa lwa <abs. end> / t v  (binary input mode)
     type is  cal  cft  civic  or  ddt
     <> shows options. keep in order. no comma for dropouts.
     nocopy.  suppresses non-instruction lines
     novla.   defeats automatic setting of vl and a registers
     mboff.   suppresses mem bank conflict checking
     +x.      assumes both +. and x. (increases mem bank checking)
     +.       assumes present routine is loaded on a 20b boundary
     x.       assumes externals are loaded on 20b-word boundaries
     +xreloc. oct  sets both +reloc. and xreloc. (affects i-buff chks)
     +reloc. oct  =>offset=oct for local word 0 in i-buff and mem bank
     xreloc. oct  =>offset=oct for external reloc vars and subrs
     iboff.   suppresses instruction buffer checking
     noiclr.  suppress clearing an i-buff after out-buf uncond jmp
     &        to continue execute line
     fwa,lwa  are octal; may have a,b,pa,pb,etc. parcel tags
     abs.     changes assumed 3400b minus word offset to 0
     end      says don't ask for additional  fwa lwa  pairs
     outfile name will be  h+infile name
   type  delaycd  for list of delay codes
   type  helpcc   for list of infile control card options


HELPCC:

     in col 1 of cycles' input file (cal,civic,cft,ddt) use:
   ln         to set vector length to n (decimal)
   cn         to reset registers and set next issue time to n (decimal)
   bn         to clear n additional instruction buffers
   in         to set next issue to n (dec) without resetting registers
   am,n       to set register am to value n (decimal)
   repeat n   before target instr to time n jumps back to target
   cycle off  disable cycle counting, use after conditional jump
   cycle on   resume counting at the 'in buffer' jump time
   cycle in   same as cycle on
   cycle out  resume counting at the 'out of buffer' jump time
     use any of these as comments in your cal infile: *am,n etc.
     in TTY mode use  ln cn in an,m  as above, and use
   ploci      to set parcel to word 'loc' and parcel i=a,b,c,d,pa,
```

Table of Delay Codes

```
DELAYCD:

  octal delay codes:
     1b functional unit not ready
     2b result register not ready
     4b operand register not ready
    10b waiting for chain cycle
    20b a or s register trunk conflict
    40b scalar memory operation bank conflict
   100b conditional jump delayed by a0 or s0 busy last 2 cycles
   200b instruction buffer delay
   400b operand chain cycles don't match. can't chain.
  1000b missed chain cycle
  2000b waits for all instructions to complete
  4000b waiting for register block transfer to finish
 10000b instruction fetch delayed by memory busy
 20000b current instr in unexpected buffer. no delay added
 40000b possible two parcel split delay of 17c
```

V. EXAMPLES

As mentioned above, source code for all these examples is in public LIB file CLASS on the CRAY-1. The timing numbers are from code CYCLES.

ZVSEEK

ZVSEEK is a BASELIB function designed to find a target value in an unordered list. The original version was written about a year before the LLNL machine arrived and has since been upgraded by use of timing analysis to run more than twice as fast. Most of the speed increase was obtained through a simple algorithm change: replacement of a logical vector instruction by a fixed add. However, an additional healthy gain came through improved handling of the vector looping technique. The main loop of the original routine consists of 10 instructions.

Main Loop of ZVSEEK (Old Version).

This version prestores the target at the end of the search array, so that it must eventually exit on a hit.

Timing of original version: VL = 64.

Address	Instruction		I	C	O	F	R	Comment
L64	V0	,A0,1	0	9	-	68	73	Get next 64 values
	V1	S4\V0	9	13	73	77	77	XOR each with target
	VM	V1,Z	77a	-b	141	145	147c	Check for hit
	S1	VM	147c	148				VM to S for count
	S0	VM	148	149				VM to S for test
	A4	ZS1	149	152				Count left zeroes (needed if hit)
	JSN	HIT	151d	156				Exit if hit
	A0	A5+A6	153	155				LOC. of next 64 values
	A5	A5+A6	154	156				Up A5 by 64
	J	L64	155	160e				Go check next 64 values

Notes:

a. Since the VM is set by the logical functional unit, this instruction, which also uses the logical unit, delays until the unit is free and does not chain.

b. The vector mask instruction never chains its output to anything.

c. While another logical vector operation using the VM-register could start at cycle 145 (for example, merge), the VM cannot be read out to an S-register until two cycles later (see the CRAY-1 Hardware Manual, p. 4-69 or page 122 of the online version, LCSD-158). Thus, we record 147 as the register free cycle.

d. This instruction is delayed one cycle since S0 has not been ready for the necessary unused cycle.

e. As written, this loop is taking 160 cycles for each 64 elements searched.

Improved Version with XOR Rep'aced by Fixed Subtract

Address	Instruction		I	C	Ø	F	R	Comments
L64	V0	,A0,1	0	9	-	68	73	
	A0	A5+A6	1	3				No reason to wait
	V1	S4-V0	9	14	73	77	78	Subtract each from target
	A5	A5+A6	10	12				Get it out of the way
	VM	V1,Z	14f	-	78	82	84	
	S0	VM	84g	85				
	S1	VM	85	86				
	A4	ZS1	86	89				
	JSN	HIT	87	92				
	J	L64	89	94h				

Notes:

f. Since the fixed subtract was used in place of the logical difference, the vector mask instruction can now chain its input operands.

g. Exchanging the order of the VM transmits to S saves a cycle later on.

h. The loop is now performing the same service as before but using only 94 cycles for each 64 elements searched.

This latter loop represents approximately a 40% improvement over the former. However, because: (1) no functional unit is used for more than 68 cycles, (2) no register is used for more than 73 cycles, and (3) there are plenty of unused registers, one would expect that additional savings may be possible.

Another item that should be taken into consideration is that this method is rather inefficient for those searches in which the target value is found

in the first portion of a set of 64 elements searched. For example, suppose the list we are searching has 64 entries. On the average, we would expect to find the target value in the first half of the list as often as in the last half, but for all these cases, the loop as written will require the full list to be tested.

In fact, there is a clever (almost heroic) method available which can go through this particular search loop in exactly 68 cycles per 64 elements searched. The treatment below, however, is somewhat easier to code (and debug) and offers an improvement in the time used to find the target over even the heroic method, on the average, for searches up to 512 in length.

The main tricks employed are: (1) breaking the array into vectors of length 32 each; (2) replicating the loop but using a different set of V-registers for each half, (3) loading and subtracting a second set of 32 elements while waiting for the VM instruction for the first 32 to finish, and (4) loading extra unneeded elements in the first half of the loop and using an otherwise unneeded vector operation in the second half to maintain the correct timing so that the load-subtract-VM chain will not be broken.

The timing chart for the main loop is given below. The notes following are referenced by line number.

	Address	Instruction		I	C	Ø	F	R
1	First half of main loop							
2								
3		A2	35					
4		A5	ADDRESS					
5		S4	TARGET,					
6								
7	L64	A0	A5	0	2			
8		VL	A2	1	2			
9		V0	,A0,1	2	11	-	41	46
10		A6	32	3	4			
11		S6	A6	4	6			
12		VL	A6	5	6			
13		A6	A4	6	8			
14		V1	S4-V0	11	16	43	47	48
15		S1	VM	15	16			
16		VM	V1,Z	16	-	48	52	54
17		S0	S1	17	18			
18		A4	ZS1	18	21			
19		JSN	HIT	20	25			
20		S0	S6-S3	22	25			
21		A6	32	23	24			
22		A5	A5+A6	24	26			
23		S3	S3+S6	25	28			
24		JSP	DUN	27	32			
25								

26	Second half of main loop							
27								
28		A0	A5	29	32			
29		V2	,A0,1	41	50	-	77	82
30		V3	S4-V2	50	55	82	86	87
31		A4	15	51	52			
32		S1	VM	54	55			
33		VM	V3,Z	55	-	87	91	93
34		S0	S1	56	57			
35		VL	A4	57	58			
36		A4	ZS1	58	61			
37		V0	V6<A0	59	65	74	78	80
38		JSN	HIT	60	65			
39		S0	S6-S3	62	65			
40		A5	A5+A6	63	65			
41		S3	S3-S6	64	67			
42		A4	32	65	66			
43		JSM	L64	67	72			
44								
45	DUN	A5	A5-A6	69	71			

Notes:

Line 8. Although we are only going to check 32 elements, we take care to load 35. The reason for this will appear at line 33.

line 9. Since there are 35 elements being loaded, F = 2+35+4.

Line 12. Now we cut the VL back to 32. Reducing the vector length in the middle of a chain is perfectly safe. However, increasing it while chaining can lead to wrong answers (i.e., the answers may differ depending on external happenings such as I/O activity, system interrupts, and operands out of range).

Line 16. The chain continues, with the functional unit becoming free at cycle 52, while the VM itself is not transmittable to S1 until 54.

Line 29. When we reach here we are simply waiting for the previous vector mask instruction at line 16 to finish. Since the memory functional unit is free, we may as well start to load the next 32 elements. We choose not to load 35 elements this time.

Line 30. The fixed adder is also free so we may as well start the next subtract at chain time.

Line 32. We must rescue the previous VM register setting before we can form a new one. Cycle 54 is the earliest this can be done.

Line 33. The cycle following the move of the VM to S1 is the first cycle in which we can start a new VM instruction. Happily, cycle 55 is also

the chain cycle for the subtract at line 30, so the chaining continues. Notice what would have happened if we had loaded only 32 elements at line 9. First, for that instruction, the functional unit would have gone free at cycle 38. Second, the load at line 29 would have then begun at cycle 38. Third, the subtract at line 30 would have chained at cycle 47. Finally, the VM at line 33 would have missed the chain cycle (52), since we had to hold it up for the move of the old VM to S1. Thus, it would not have issued till cycle 87.
But since the loop will normally continue back to the VM at line 16, and since we have not loaded 35 elements this time, we must do something to hold back the load at line 9 in the next pass, or the VM at line 16 will again miss its chain cycle.

Line 35. Here we start to pull another trick, which will delay the load at line 9 in the next pass and at the same time protect this loop against a problem (in timing, not correctness) that may arise if there is an interrupt during its execution. The protection is free in terms of the cycles required to do it, but it does require extra instructions.

Line 37. This is the protection instruction. Since it is putting 15 results into V0 using the shift functional unit, which has a chain time of 6, it will tie up register V0 until cycle 80. This in turn will cause the next load at line 9, which uses V0, to be held until cycle 80. This is the exact cycle desired, since it will bring the chain cycle from the subtract at line 14 to cycle 94, the cycle immediately after the one in which we can first save the VM (93). At the same time, regardless of whether or not some interrupt has come along and bollixed our careful timing, this will force the next load (at line 9) to hold long enough relative to the previous VM so that we will be back in synch thereafter.

Line 38. In this program address HIT has already been put into an instruction buffer. If this were not the case, the jump would complete at cycle 91.

Lines 37 through 40.
Several instructions are completing in cycle 65; each uses a different register set.

Line 43. After jumping back, we will be holding at line 9 for the completion of the instruction at line 37. The loop time will be 78 cycles for each 64 elements tested, but, on the average, we will exit in the upper half of the loop half the time, which provides a further speed increase, especially valuable for short arrays.

QVDIVO

As another example, we present the coding for QVDIVO, the CRAY-1 STACKLIB divide routine.

On the CRAY, the vector divide algorithm used to accomplish the FORTRAN vector statement C = A/B, where A, B, and C are vectors with arbitrary (linear) stride, requires three vector memory operations, three vector multiply operations, and one vector reciprocal approximation instruction for each 64 elements. The current CFT implementation of the general vector divide loop requires 445 cycles per 64 elements stored plus some startup time, which brings the cost for such a divide to roughly 7 cycles per element. However, by overlaying the storing of the result for the first pass through the loop and the loading of the operands for the third pass through the loop with the multiplying still being carried out for the second pass, one can expect to achieve something on the order of twice CFT's performance. In fact, the theoretical minimum, 205 cycles (68 + 68 for loads + 69 for store) per 64 elements (after suitable startup time) is achieved in this routine. The timing chart for the main loop is given with notes below.

Line		Instruction		I	C	O	F	R
-3		V6	V2*IV1	-137	-128	-73	-69	-64
-2		V4	V1*FV6	-64	-55	0	4	9
-1		A0	S5	-63	-62			
0		JSP	TWOTRIP	-62	-48			
*					B U F F E R	B O U N D A R Y		
1		V2	,A0,A5	-48	-39	16	20	25
2	LP	VL	A4	-44	-43			
3		A3	A5*A7	-43	-37			
4		S3	A2	-42	-40			
5		V1	S0+V5	0	5	64	68	69
6		S3	S3<6	1	3			
7		S2	S3+S2	3	6			
8		V6	V7*IV1	5	14	69	73	78
9		S3	A3	6	8			
10		S5	S5+S3	8	11			
11		A0	S5	11	12			
12		V0	,A0,A5	20	29	-	88	93
13		VL	A7	21	22			
14		V3	V4*RV2	73	82	137	141	146
15		VL	A4	74	75			
16		A0	S2	75	76			
17		V7	,A0,A2	88	97	-	156	161
18		V5	/HV7	97	113	161	165	177
19		V2	V0&V0	137	141	201	205	205
20		V4	V1*FV6	141	150	205	209	214
21		VL	A7	142	143			
22		A0	S6	143	144			

23	A3	A6*A7	144	150			
24	,A0,A6	V3	156	-	220	225	-
25	S3	A3	157	159			
26	S0	S1-S4	158	161			
27	S1	S1-S4	159	162			
28	A7	A4	160	162			
29	S7	S4	162	163			
30	S6	S6+S3	163	169			
31	JSN	LP	164	169			

Notes:

Line -3. We choose to begin the timing chart somewhat before the loop. We have to start the timing somewhere. Arbitrarily, we may take the start of this instruction as any cycle. Cycle -137 will be convenient.

Line -2. At this point, it is clear that the state of the machine prior to line -3 will have no effect on the issue time of this instruction. (Actually, a vector reciprocal instruction whose result register was V4 could still be in progress and would delay this issue by a few cycles, but that is not the case.)

Lines 0 and 1.
The jump here to TWØTRIP is not taken. However, a 16-word buffer boundary (20 octal) occurs after the JSP instruction, and this delays the next instruction until the new buffer can be loaded from memory. Notice that the time of issue of the instruction at line 1 after the buffer load is the same as it would have been had a jump been taken to it.

Line 5. This move instruction is the first vector instruction in the loop. We have arranged to make it issue at cycle 0. It will wait to issue until V1 has delivered all the operands for the multiply instruction at line -2.

Line 8. This multiply chains with the fixed add (move) at line 5. We have insured chaining by delaying the move long enough to have the multiply functional unit free from line -2.

Line 12. This load will issue as soon as the previous one at line 1 releases the memory (cycle 20).

Line 14. V2, V3, and V4 have been available for many cycles before this instruction can issue. It has to wait for the use of the multiply unit. Note also that the A-register multiplies do not interfere with the floating-point multiplies since they are done in a separate functional unit.

Lines 17 and 18.

These instructions chain.

Line 19. This is another move instruction. The release of V0 by this instruction determines the length of the loop (205 cycles).

Line 20. This does not chain with the move at Line 19. It issues at cycle 141 because it can't get at the multiply unit from line 14 before then.

Line 24. This is the final store instruction. It is released for issue by the availability of the memory from line 17. The memory functional unit also determines the time for the loop since we are using it for 68+68+69=205 cycles.

The following is a timing and accuracy test for QVDIV0:

```
*       RCFT I=TESTQVD,ON=G,B=BQVD,C=COO
* LDR I=(BQVDIV,BQVD),X=XQVD,ORDER=CLNB,FIRST=BQVDIV
*  XQVD
      COMMON /QVCOM/ X(48000),W(48000),U(48000),Z(48000)
      CALL LINK('UNIT59=(TTY,TEST)//')
      DO 3 L = 1,12000,64
      DO 2 I=1,3*L+1
      Z(I) = 4+L
    2 U(I) = 4+1
      K = IRTC(0)
      CALL QVDIV0(W(1),U(1),Z(2),L,4,3,2)
      N = IRTC(0)
      N = N-K
      K = IRTC(0)
      DO 1 I=0,L-1
      X(4*I+1) = U(3*I+1)/Z(2*I+2)
    1 CONTINUE
      M = IRTC(0)
      M = M-K
      DO 4 I=0,L-1
      IF(X*I+1).NE.W(4*I+1)) GO TO 5
    4 CONTINUE
      WRITE(59,60) L,M,N
   60 FORMAT(16,216)
    3 CONTINUE
      STOP 1
    5 CONTINUE
      WRITE(59,59) W(4*I+1),X(4*I+1)
      WRITE(59,61) (W(I),I=1,4*L-3,4),(X(I),I=1,4*L-3,4)
   59 FORMAT(3F18.14)
   61 FORMAT(3022)
      STOP
      END
```

QVSQRTH

We conclude our examples with the code for QVSQRTH, a half precise (28-bit-accurate) square root routine for arrays, available in STACKLIB. The full-precision routine QVSQRT is quite similar, requiring one additional iteration but needing, also, a full precision divide during this final iteration. The code is perhaps remarkable in that maximum speed is obtained by breaking the array up into vectors of length 31, and because every vector operation is chained to the previous one. A total of 21 consecutive chained vector operations occur.

Essentially, the idea is to compute an initial guess X0 and then to iterate three times by the formula: Xi+1 = (Xi + Y/Xi)/2, where Y is the number whose square root is desired. The iterative loop can be managed by the four CAL instructions:

```
V0  /HV1
V2 V0*FV3
V4 V2+FV1
V5 S4+V4
```

The halving operation is performed by adding minus one to the exponent. Chaining will end for long vectors at the (*F) instruction since there will be a conflict over the use of register V1. However, by adding one auxiliary NO-OP instruction (a shift of zero), we can achieve the following timing for vectors of length 31, since the +F is delayed until V1 is free.

	I	C	Ö	F	R
V0 /HV1	0	16	31	35	47
V6 V0*FV3	16	25	47	51	56
V2 V6>A7	25	31	56	60	62
V4 V2+FV1	31	39	62	66	70
V5 S4+V4	39	43	70	74	75

Now, at cycle 43, we can issue another reciprocal operation (to register V7) and continue the procedure without any breaks in the chain. Moreover, since the initial guess can be generated by a similar set of chained operations, the entire calculation may proceed from the initial load, with each successive vector instruction issuing at the chain cycle of the previous one. (In the full-precision routine, the chain is broken during the calculation of the full-precision reciprocal.)

The timing chart for this half-precise square root is given next (for the main loop). A full iteration begins at label ITER. The complete routine is available in file CLASS.

Address	Instruction		I	C	Ø	F	R
LØØP	V5	V0*FV1	0	9	31	35	40
	V6	V5>A7	9	15	40	44	46
	A7	24	10	11			
	A7	V6+FV7	15	23	46	60	54
	A0	A1+A7	17	19			
	A7	A7-A6	18	20			
	V3	S5+V2	23	28	54	58	59
	S0	+A7	24	26			
	A7	-A7	25	27			
	S7	VM	26	27			
	V4	/HV3	28	44	59	63	75
	JSP	NØLØD	29	34			
	VL	A7	31	32			
	V0	,A0,A3	32	41	39	43	48
	VM	V0,Z	41	52	48	52	54
	VL	A6	42	43			
NØLØD	V5	V4*FV1	44	53	75	79	84
	A7	0	45	46			
	V0	V5>A7	53	59	84	88	90
	S7	S6&S7	54	55			
	A7	A6*A3	55	61			
	S2	VM	56	57			
	V6	V0+FV3	59	67	90	94	98
	S2	S2>24	60	62			
	S7	S2!S7	62	63			
	VM	S7	63	66			
	A1	A1+A7	64	66			
	V2	S4!V6&VM	67	71	98	102	102
	A0	A6-A5	68	70			
	A7	A6*A4	69	75			
	A5	A5-A6	70	72			
	V7	S5+V2	71	76	102	106	107
	JAP	DUN	72	77			
	A0	A6-A5	74	76			
	JAP	SHØRT2	78	83			
ITER	A1	A1	80	82			
	A0	A1	82	84			
	V1	,A0,A3	84	93	115	119	124
RTN2	V0	S1*FV1	93	102	124	128	133
	S2	>2	94	95			
	S2	S2>15	95	97			
	V2	S2!V0	102	106	133	137	137
	V3	V2>A0	106	112	137	141	143
	A0	A2	107	109			
	A2	A2+A7	108	110			
	V4	S3+V3	112	117	143	147	148
	V5	/HV4	117	133	148	152	164
	,A0,A4	V7	118	-	149	154	-

V6	V5*FV1	133	142	164	168	173
A7	24	134	135			
VL	A7	135	136			
VM	V0,Z	137	165	161	165	167
VL	A6	138	139			
A7	0	140	141			
V2	V6>A7	142	148	173	177	179
V3	V2+FV4	148	156	179	183	187
V7	S5+V3	156	161	187	191	192
V0	/HV7	161	177	192	196	208
J	LOOP	162	167			

APPENDIX A. AN ABRIDGEMENT OF THE SUMMARY OF CPU TIMING INFORMATION FURNISHED BY CRAY RESEARCH INC.

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 scalar mask instruction 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. An asterisk indicates that issue may be delayed because of a functional unit reservation by a vector instruction. Memory may be considered a functional unit for timing considerations.

(A=A-register, M=Memory, B=B-register, S=S-register, I=Immediate,
C=Channel, T=T-register, V=V-register, * see previous page)

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<--(r.a.)S	14*
S<--I	1	S<--V	5
S<-S(log)S	1	V<--S	1
S<-S(shift)I	2	S<--VM	1
S<-S(shift)	3	S<--RTC	1
S<--S(mask)	1	S<--A	2
RTC<--S	1	VM<--S	3

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.

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).NE.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 the A0 or S0 operand register has been free for two clock periods. 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. (18 clock periods for 8-bank phasing option.)

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 12 clock periods if the next instruction parcel is not in an instruction parcel buffer.

---

* Multiple of 4 for 8 bank phasing option.

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.

Scalar load 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+10    Clear register reservation
CP n+11    Complete and issue waiting instruction
```

APPENDIX B. WHAT HAPPENS WHEN YOU RUN ON THE CRAY?

You type your LOGON at a terminal; then: (*)

1. The LOGON line goes to the COMBO checker, which verifies it, appends some bits of information and sends it on.

2. The line next arrives at the TMDS concentrator, which notes that it is destined for the CRAY and routes it to the A410.

3. The A410 performs the appropriate protocol and drops the line onto the NSC bus.

4. The A130, which is attached to a CRAY channel, picks up the line from the bus and sends it along the CRAY channel to an LTSS memory buffer.

5. LTSS, which is frequently polling all CRAY channels, notices the activity, sees that this is a LOGON line, and verifies that you are an authorized user.

6. LTSS then prepares an index of private and public disk files to which you have access and associates it with your user number.

7. LTSS returns an appropriate acknowledgment of your LOGON and sends it on the reverse route to your teletype.

The acknowledgment response and all subsequent message lines bypass the COMBO checker. In fact, if the COMBO checker was down at initial LOGON time, the LOGON line would go directly to the TMDS concentrator.

---

* For the MFE network, replace items 1 through 4 above by the following:

M1. The LOGON line goes via a modem and telephone lines to a VADIC modem multiplexor, which sends it on (or it may go directly to step M2).

M2. A PDP-11 concentrator then notes that it is destined for the CRAY and routes the line to a 7600 PPU (12). (In the future, another PDP-11 will be used.)

M3. The PPU performs the necessary protocol and sends the lines to the CRAY-7600 Adaptor.

M4. The adaptor, which is attached to a CRAY Channel, picks up the line and sends it along to a CTSS memory buffer.

Next, you type in an EXECUTE line, say, CLASS / 1 .7, which goes to CRAY LTSS,

1. A search is made of your private file index to determine whether you have a file by the name of CLASS.

2. If not, a search is made of your PUBLIC file index to see if it has a file by that name.

3. If not, the message "NO FILE" is sent to your terminal.

4. When CLASS is found, your PRIORITY is checked (V/TL = .7), and if necessary, changed to conform to the current limits, or, if your account has no time left, changed to S (standby).

5. The job is then assigned to an appropriate loading queue and, when memory space is available, a number of words equal to the load length of this file is brought into memory.

6. When the file is in memory, LTSS performs a sequence of validity checks on the minus words. If any check fails, an appropriate message is returned to your terminal, and execution ceases.

7. If all seems well, the job is placed in an appropriate queue and scheduled for CPU time.

8. When the proper time arrives, LTSS relinquishes control of the CRAY CPU to your program by exchanging from MONITOR to JOB mode, putting the contents of your minus words into the CRAY registers, and requesting the 16-word buffer-load of instructions containing the instruction addressed by your program counter to be fetched to an instruction buffer.

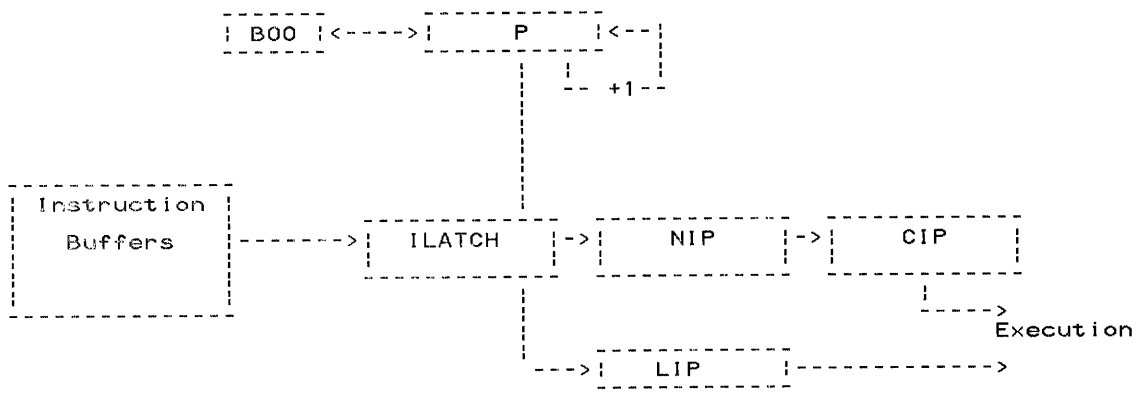
9. Finally, then, the first instruction will be performed and the program counter advanced to the next instruction.

10. In general, your program continues in control of the CPU until it makes a recognized error, gives control back to LTSS, or is interrupted by LTSS. However, while it is in control of the CPU, LTSS may have on-going I/O activity, which will share the use of memory with your program.

APPENDIX C. THE DETAILS OF INSTRUCTION FETCH TIMING

All this detail is incorporated in the code CYCLES.

There are essentially five registers to consider, a few flags and a few time positions.



An instruction which issues at cycle x must have entered the CIP at cycle x-1 or before, the NIP at cycle x-2 or before, and the ILATCH at x-3 or before. Some time prior to cycle x-3, the instruction must have been located in one of the four 64-parcel instruction buffers, and before that, it was in memory.

In general, instructions coming from the instruction buffers are able to reach the CIP at a rate of one per cycle; however, when the end of a buffer is reached, delays are encountered in locating the next instruction to be processed. Similarly, whenever Branch instructions cause the orderly flow of sequential instructions to be interrupted, delays are to be expected.

The chart (pages 60-61) illustrates details of the flow of instruction parcels in the CRAY-1. Registers involved in this flow are described in the "Instruction Issue and Control" section of Chapter 3 of the CRAY Hardware Reference Manual.

In general, the P register is incremented by one each time an instruction is issued. If the instruction parcel corresponding to the new P value in sequence is in the current instruction buffer, then that parcel goes

to the ILATCH register during the same cycle. If the parcel is not in the current I-buffer, then the ILATCH INVALID flag is set.

If the required parcel is not in any I-buffer, then a memory instruction fetch request (IFR) is issued. Normally, four instruction words (16 parcels) including the required parcel will arrive in the next I-buffer eleven cycles after the IFR. If memory is already busy then the IFR must wait. The other twelve instruction words to fill the I-buffer will be requested in groups of four during the next three cycles. The required parcel reaches ILATCH in the same cycle it reaches the I-buffer. I-buffers are loaded in strict rotation regardless of when the buffer was used last.

If the required parcel is already in a different I-buffer, then CHANGE BUFFER is set and on the following cycle the current I-buffer designator is switched. The correct parcel will reach ILATCH on the following cycle, two cycles delayed. A jump within the current I-buffer takes as long as a jump to a different I-buffer.

An instruction issues from the CIP (current instruction parcel) register. The second parcel of a two-parcel instruction issues from the LIP (lower instruction parcel) register. In the same cycle a new parcel moves into CIP from the NIP (next instruction parcel) register unless blocked by the TPS (two parcel split) flag. The TPS flag is set when ILATCH is invalid and NIP contains the first parcel of a two parcel instruction. (17d)

In the same cycle that a parcel moves from NIP to CIP, a parcel moves from ILATCH to NIP unless blocked by the ILATCH INVALID flag described above. If NIP contained the first parcel of a two parcel instruction, then the parcel in ILATCH goes to LIP instead, and a NOP is placed in NIP.

With these rules we are now ready to use the chart below which illustrates the cycle-by-cycle progress of instruction parcels for the following code sequence:

```
addr    parcel    CAL mnemonics

 17a    072700    s7  rt
 17b    020100    a1  two
 17c    000002
                 *repeat 1
 17d    031110    a1  a1-1
 20a    030001    a0  a1
 20b    011000    jan *-2
 20c    000077
 20d    072600    s6  rt
 21a    004000    ex
                two = 2
```

Assume that completion of an exchange sequence results in setting the P register to 17a in cycle 1.

IFR means "instruction fetch request" issued for these words. x column shows nip entry blocked because invalid data in ilatch. - means invalid or irrelevant data.

cycle	IFR words	words ready	p reg	ilatch	x	nip	(lip)	cip	instr. issued	comments
1	14-17	-	17a	-	x	-	-	-	-	IFR for 14a-17d
2	0- 3	-	17a	-	x	-	-	-	-	(ready in I-buffer
3	4- 7	-	17a	-	x	-	-	-	-	11 cycles after
4	10-13	-	17a	-	x	-	-	-	-	memory request)
5	-	-	17a	-	x	-	-	-	-	waiting
6	-	-	17a	-	x	-	-	-	-	for
7	-	-	17a	-	x	-	-	-	-	instructions
8	-	-	17a	-	x	-	-	-	-	to
9	-	-	17a	-	x	-	-	-	-	arrive
10	-	-	17a	-	x	-	-	-	-	from
11	-	-	17a	-	x	-	-	-	-	memory
12	-	14-17	17a	17a		-	-	-	-	11 cycles after IFR
13	-	0- 3	17b	17b		17a	-	-	-	
14	-	4- 7	17c	17c		17b	-	17a	-	
15	-	10-13	17d	17d		nop	(17c)	17b	17a	s7 = rtc at this cycle
16	20-23	-	20a	-	x	17d	-	nop	17b	IFR for 20a-23d
17	24-27	-	20a	-	x	-	-	17d	nop	a1 now set to 2
18	30-33	-	20a	-	x	-	-	-	17d	a1-1 to address adder
19	34-37	-	20a	-	x	-	-	-	-	
20	-	-	17a	-	x	-	-	-	-	a1 now set to 1
21	-	-	17a	-	x	-	-	-	-	waiting for
22	-	-	17a	-	x	-	-	-	-	instructions
23	-	-	17a	-	x	-	-	-	-	to
24	-	-	17a	-	x	-	-	-	-	arrive
25	-	-	17a	-	x	-	-	-	-	from
26	-	-	17a	-	x	-	-	-	-	memory
27	-	20-23	20a	20a		-	-	-	-	11 cycles after IFR
28	-	24-27	20b	20b		20a	-	-	-	
29	-	30-33	20c	20c		20b	-	20a	-	
30	-	34-37	20d	20d		nop	(20c)	20b	20a	0+a1 to address adder
31	-	-	20d	20d		nop	(20c)	20b	-	
32	-	-	20d	20d		nop	(20c)	20b	-	a0 ready (=1)
33	-	-	20d	20d		nop	(20c)	20b	-	a-branch flags set
34	-	-	17d	-	x	-	-	nop	20b	17d goes to p-counter
35	-	-	17d	-	x	-	-	-	nop	
36	-	-	17d	17d		-	-	-	-	
37	-	-	20a	-	x	17d	-	-	-	change buffer request
38	-	-	20a	-	x	-	-	17d	-	
39	-	-	20a	20a		-	-	-	17d	a1-1 to address adder
40	-	-	20b	20b		20a	-	-	-	

```
 41     -      -     20c    20c      20b     -      20a     -
 42     -      -     20d    20d      nop   (20c)    20b    20a    0+a1 to address adder
 43     -      -     20d    20d      nop   (20c)    20b     -
 44     -      -     20d    20d      nop   (20c)    20b     -     a0 ready (=0)
 45     -      -     20d    20d      nop   (20c)    20b     -     a-branch flags set
 46     -      -     21a    21a      20d     -      nop    20b    (drop through)
 47     -      -     21b    21b      21a     -      20d    nop
 48     -      -     21c    21c      21b     -      21a    20d    s6 = rtc = s7+33
 49     -      -     21d    21d      21c     -      21b    21a    exit
```

```
cycle   notes
  1   words 14-17 are requested from memory.
 12   words 14-17 reach I-buffer 0 and parcel 17a enters ILATCH.
 15   parcel 17a issues fourteen cycles after being requested from memory.
 16   17b issues and parcel 20a (words 20-23) is requested from memory.
      In general, the next buffer is requested when 17b issues from the
      old buffer.  If 20a is not in an I-buffer then it will be ready to
      issue after fourteen more cycles, unless further delayed by memory busy.
 30   parcel 20a issues fourteen cycles after 17b issued and IFR.
 32   register a0=0+a1 is ready.  The result is sent to the A0 branch flag
      setting unit.  This would not delay instructions other than jump
      on A0 instructions.
 33   the A0 branch flags are set.
 34   now the Jump on A0 Non-zero can issue which resets the P register.
      A jump to a parcel already in an I-buffer takes 5 cycles for the target
      parcel to issue.
 37   parcel 20a is requested when 17d leaves ILATCH.  20a is in an I-buffer
      and will be in ILATCH in two cycles.
 39   target parcel 17d issues and 20a reaches ILATCH.
 42   parcel 20a issues as in cycle 30.
 46   JAN issues but this time the P register is not reset and we drop
      through.
 48   the real-time clock reading would be 33 cycles greater than cycle 15.

      CYCLES' output for this code sequence:

  loc   instr            res   operand      w b delay    i    c    o    f    r

00017a 072700           s7    rt             A20000    15   16                  
00017b 0201 00000002    a1    two                      16   17                  2=a1
00017d 031110           a1    a1-1                     18   20                  1=a1
00020a 030001           a0    a1           11B00204    30   32                  1=a0
00020b 011 00000017d    jan   17d           3 00100    34   39    a    39    48

     jump back to repeat at     17d

00017d 031110           a1    a1-1            a        39   41                  0=a1
00020a 030001           a0    a1           2b00204     42   44                  0=a0
00020b 011 00000017d    jan   17d           3 00100    46   51    a    51    60
00020d 072600           s6    rt                       48   49
000210 004000           ex                    1 02000   50  100
```

TTY input to CYCLES for this example:

```
cycles tty htty.
p17a c15 72700 20100 2 31110 30001 11000 77
p17a 31110 30001 11000 77 72600 4000 end
```

Summary

Instruction look ahead is effectively three parcels (CIP, NIP, and ILATCH). When instruction 17b of a buffer is issued, the first parcel (20a) of the next I-buffer load is sought. If parcel 20a is already in an I-buffer then it is delayed only 2 cycles; if it is not in a buffer, then it should be ready to issue fourteen cycles after it was requested (ie. after 17b issued). The request is delayed until memory is not busy. After the request is accepted memory is busy for six additional cycles.

There are four exceptional cases to consider:

1. If 17c is a branch instruction, then the instruction fetch request (IFR) is delayed until the jump address is decided. The address is decided in the jump issue cycle except for "J Bjk" in which it is decided two cycles later.

2. If 17c is a scalar load or store which issues immediately, then it gets memory service first and the instruction fetch is delayed four cycles.

3. If 17c is a vector load or store or a block register transfer and it issues immediately, then the instruction fetch is delayed until 17c is done with memory. The delay will be VL+4 for a load and VL+5 for a store.

4. If 17c is a one parcel instruction followed by a two parcel instruction, then if 17c does not issue immediately, it will be held from issue until the second parcel of 17d reaches ILATCH. The hold is caused by the setting of the TPS (two parcel split) flag after 17d reaches NIP.

The following sequences, which differ only by the second instruction issued (at cycle 2 or 1), illustrates this effect:

```
   loc   instr            res   operand       w b delay    i     c     o     f     r

00017a 061106             s1      -s6           A          0     3
00017b 054521             s5      s5<17       1 00020      2     4
00017c 070210             s2      /hs1                     3    17
00017d 1305 00010000      10000b,0   s5       11B00200     15
00020b 064432             s4        s3*fs2                17    24
00020c 1304 00010001      10001b,0   s4        6 00004     24

   loc   instr            res   operand       w b delay    i     c     o     f     r

00017a 061106             s1      -s6           A          0     3
00017b 042521             s5      <47                      1     2
00017c 070210             s2      /hs1        11 00204     13    27
00017d 1305 00010000      10000b,0   s5         B          14
00020b 064432             s4        s3*fs2    11 00004     27    34
00020c 1304 00010001      10001b,0   s4        6 00004     34
```

In the first case, parcel 17b was delayed one cycle by an S-reg path conflict, so parcel 17c was able to issue immediately and beat the TPS hold. In the second case, parcel 17b had no trunk conflict and issued on cycle 1. Parcel 17c was, as before, ready to issue on cycle 3, but by then the TPS hold was on. Thus, 17c had to wait for 20a to reach ILATCH before the hold was released permitting it to issue.

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government thereof, and shall not be used for advertising or product endorsement purposes.

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-Eng-48.

HN/SV