CRAY RESEARCH, INC.

PRELIMINARY REFERENCE MANUAL

CRAY-1 COMPUTER

THE CRAY-1 COMPUTER

PRELIMINARY

REFERENCE MANUAL

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

FIGURES

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.

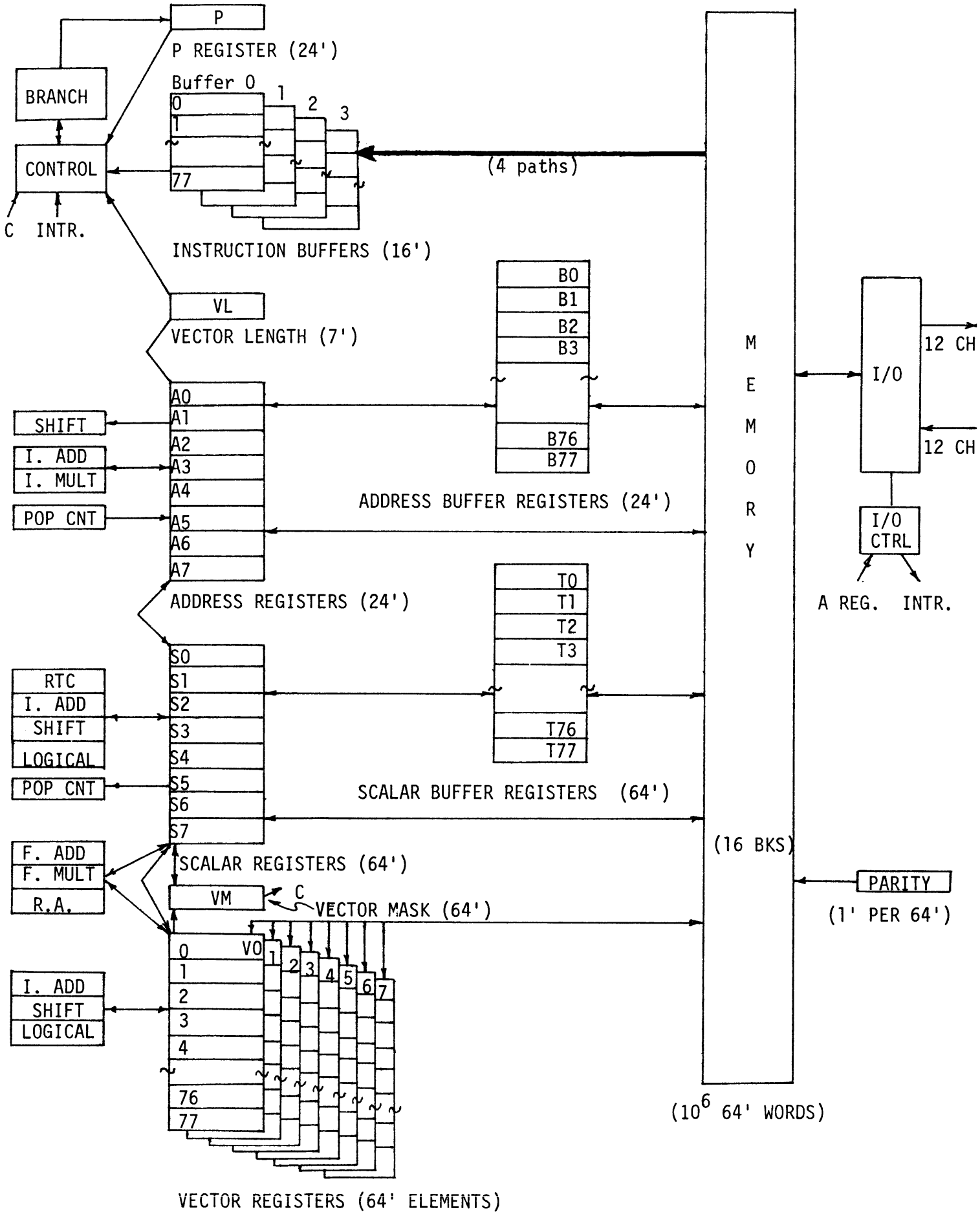
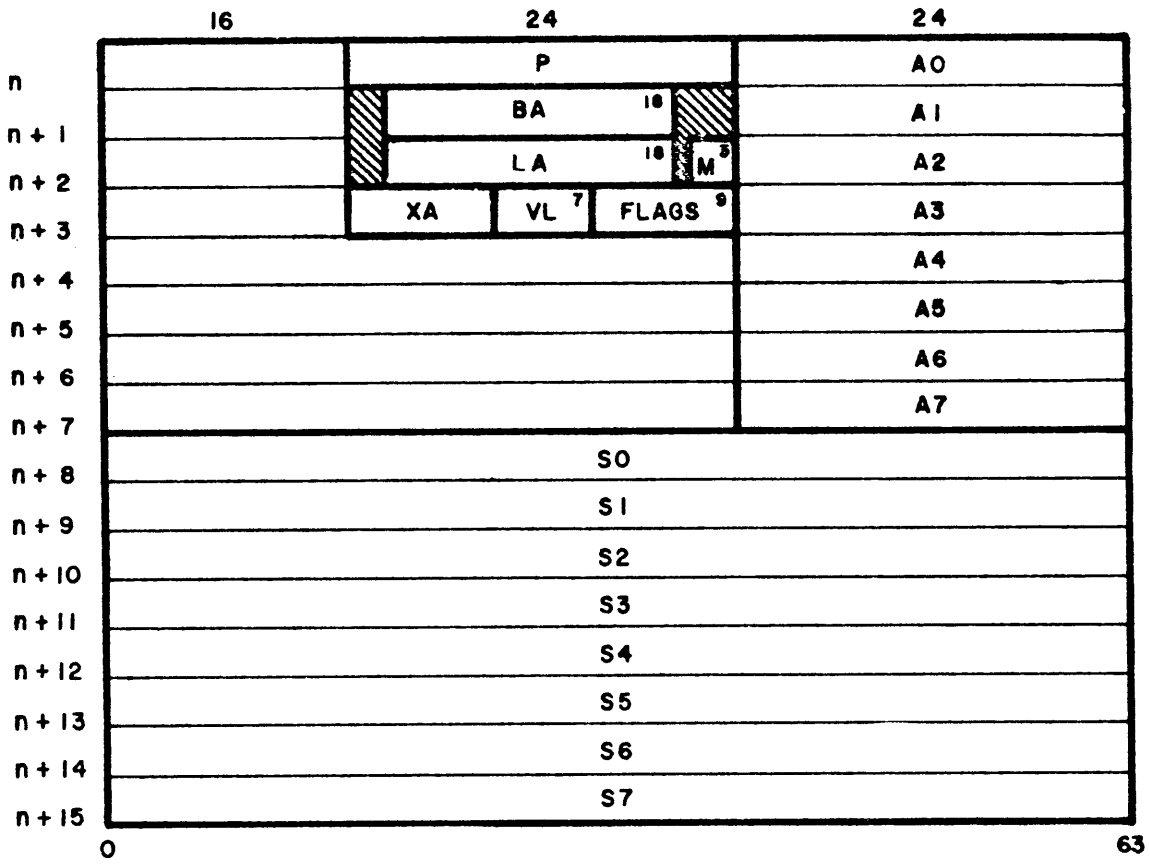


Figure 1. Registers block diagram



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.

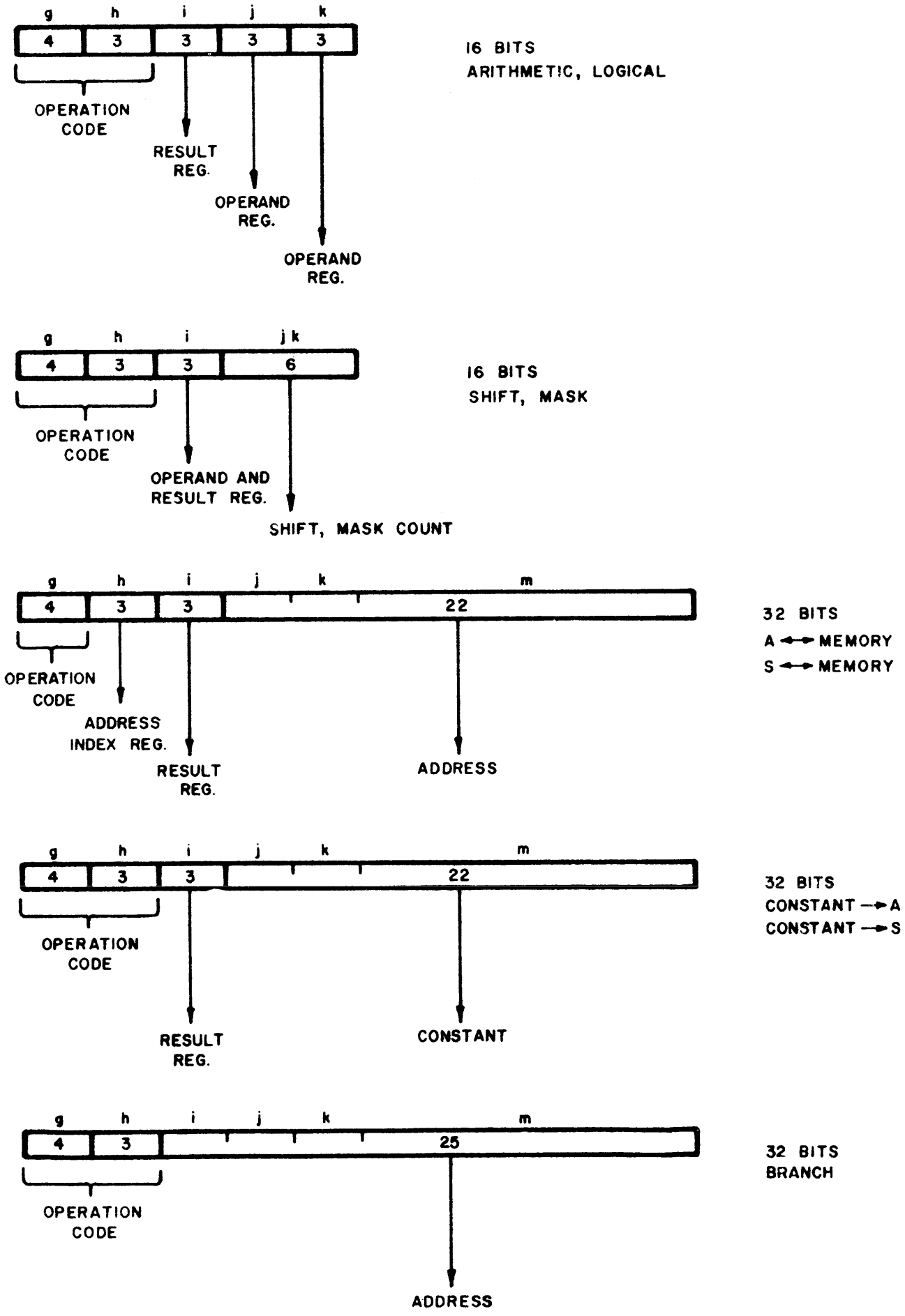


Figure 3. Instruction formats

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}

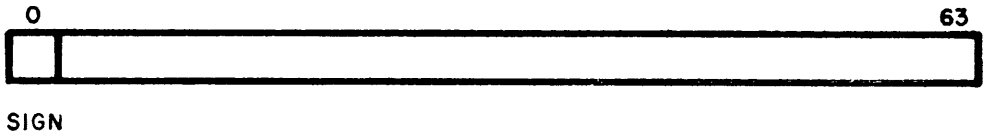
Instruction buffers

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.

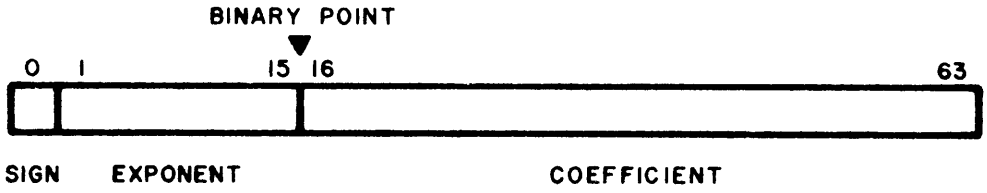
When the high order 18 bits of the P register do not match any instruction buffer base address, an "out of buffer" condition exists and instructions are read to an instruction buffer from memory. When an "out of buffer" condition occurs, the instruction buffer that receives the instruction is determined 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

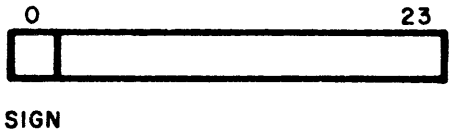
Figure 4 illustrates the data formats for integers and floating point quantities. The range for floating point quantities is $\left[10^{-5000}, 10^{2500}\right]$. Normalized floating point quantities are expressed as $z = y * 2^x$ where $y = 0$ or $\frac{1}{2} \leq y < 1$ and $-40000_8 \leq x < 20000_8$. 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