CRAY RESEARCH, INC.

CRAY X-MP AND CRAY-1® COMPUTER SYSTEMS

SYMBOLIC MACHINE INSTRUCTIONS
REFERENCE MANUAL

SR-0085

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER SR-0085

Each time this manual is revised and reprinted, all changes issued against the previous version are incorporated into the new version and the new version is assigned an alphabetic level.

Every page changed by a reprint with revision has the revision level in the lower righthand corner. Changes to part of a page are noted by a change bar in the margin directly opposite the change. A change bar in the margin opposite the page number indicates that the entire page is new. If the manual is rewritten, the revision level changes but the manual does not contain change bars.

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

CRAY RESEARCH, INC.
2520 Pilot Knob Road
Suite 310
Mendota Heights, Minnesota 55120

Revision	Description
	January 1986 - Original printing.

PREFACE

This manual provides information on CRAY X-MP and CRAY-1 Symbolic Machine Instructions, and is intended to be used as a reference with CAL Assembler Version 2.

Specific information on CAL Assembler Version 2 can be found in the following manual:

SR-2003 CAL Assembler Version 2 Reference Manual

CONTENTS

FIGURES

TABLE

INDEX

INTRODUCTION 1

Each Cray mainframe (CRAY X-MP and CRAY-1) machine instruction can be represented symbolically in Cray Assembly Language (CAL). This manual provides information on the Symbolic Machine Instructions used with the CRAY X-MP and CRAY-1.

For a general description of the Cray mainframe, refer to the appropriate Reference Manual:

- HR-0004 CRAY-1 Hardware Reference Manual
- HR-0029 CRAY-1 S Series Mainframe Reference Manual
- HR-0064 CRAY-1 M Series Mainframe Reference Manual
- HR-0088 CRAY X-MP Series Models 11, 12, and 14 Mainframe Reference Manual
- HR-0032 CRAY X-MP Series Models 22 and 24 Mainframe Reference Manual
- HR-0097 CRAY X-MP Series Model 48 Mainframe Reference Manual

Section 2 of this manual provides information on Symbolic Machine Instruction format for a 1-parcel (16-bit) instruction or a 2-parcel (32-bit) instruction. It also describes special register values that may be referenced by the instructions and the symbolic notation used for coding the machine instructions.

Section 3 provides detailed information on the CAL instructions that operate on the CRAY X-MP and CRAY-1. Each instruction begins with boxed information consisting of the CAL syntax format, an operand if required, a brief description of each instruction, and the machine instruction.

Following the boxed information is a detailed description of the instruction and an example.

Appendix A provides a summary of functional units and the symbolic machine instructions. Appendix B lists the instructions by function. References to section 3 for a detailed description of the instruction are provided.

INSTRUCTION SYNTAX 2

Each CRAY X-MP and CRAY-1 mainframe machine instruction can be represented symbolically in Cray Assembly Language (CAL). The assembler identifies a symbolic instruction according to its syntax and generates a corresponding binary machine code. An instruction is generated in the assembly section in use when the instruction is interpreted.

This section describes the format of symbolic machine instructions, special register values, and notation used for coding symbolic machine instructions for CAL Assembler Version 2 on a CRAY X-MP and CRAY-1.

2.1 INSTRUCTION FORMAT

Each instruction is either a 1-parcel (16-bit) instruction or a 2-parcel (32-bit) instruction. Instructions are packed 4 parcels per word. Parcels are numbered 0 through 3 from left to right and any parcel position can be addressed in branch instructions. A 2-parcel instruction begins in any parcel of a word and can span a word boundary. For example, a 2-parcel instruction beginning in parcel 3 of a word, ends in parcel 0 of the next word. No padding to word boundaries is required. Figure 2-1 illustrates the general form of instructions.

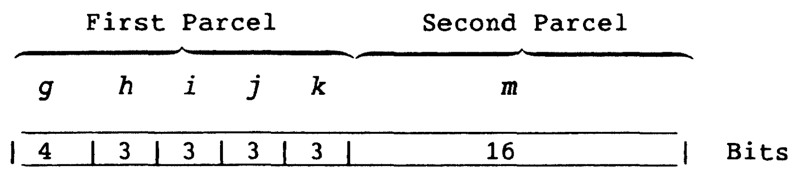


Figure 2-1. General Form for Instructions

Four variations of this general format use the fields differently. The formats of the following variations are described in this section:

- 1-parcel instruction format with discrete *j* and *k* fields
- 1-parcel instruction format with combined *j* and *k* fields

- 2-parcel instruction format with combined *j*, *k*, and *m* fields
- 2-parcel instruction format with combined *i*, *j*, *k*, and *m* fields

2.1.1 1-PARCEL INSTRUCTION FORMAT WITH DISCRETE *j* AND *k* FIELDS

The most common of the 1-parcel instruction formats uses the *i*, *j*, and *k* fields as individual designators for operand and result registers (see figure 2-2). The *g* and *h* fields define the operation code. The *i* field designates a result register and the *j* and *k* fields designate operand registers. Some instructions ignore one or more of the *i*, *j*, and *k* fields. The following types of instructions use this format:

- Arithmetic
- Logical
- Double shift
- Floating-point constant

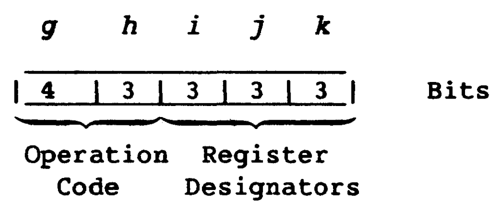


Figure 2-2. 1-parcel Instruction Format with Discrete *j* and *k* Fields

2.1.2 1-PARCEL INSTRUCTION FORMAT WITH COMBINED *j* AND *k* FIELDS

Some 1-parcel instructions use the *j* and *k* fields as a combined 6-bit field (see figure 2-3). The *g* and *h* fields contain the operation code, and the *i* field is generally a destination register. The combined *j* and *k* fields generally contain a constant or a B or T register designator. The branch instruction 005 and the following types of instructions use the 1-parcel instruction format with combined *j* and *k* fields:

- Constant
- B and T register block memory transfer
- B and T register data transfer
- Single shift
- Mask

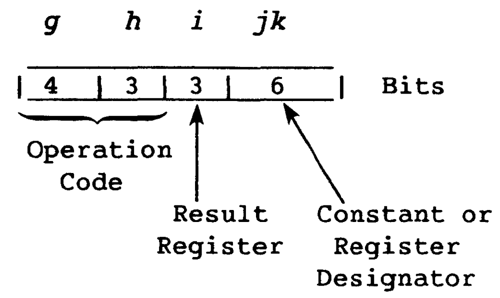


Figure 2-3. 1-parcel Instruction Format with Combined *j* and *k* Fields

2.1.3 2-PARCEL INSTRUCTION FORMAT WITH COMBINED *j*, *k*, AND *m* FIELDS

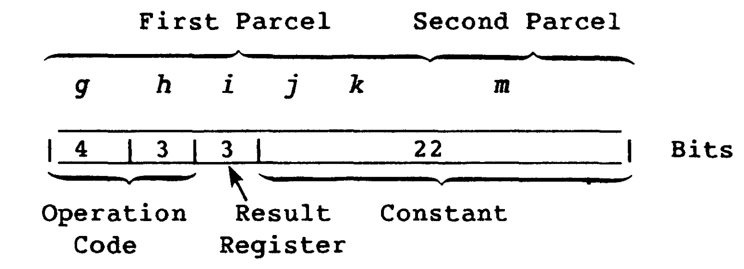
The instruction type for a 22-bit immediate constant uses the combined *j*, *k*, and *m* fields to hold the constant. The 7-bit *gh* field contains an operation code, and the 3-bit *i* field designates a result register. The instruction type using this format transfers the 22-bit *jkm* constant to an A or S register.

The instruction type used for Scalar Memory transfers also requires a 22-bit *jkm* field for an address displacement. This instruction type uses the 4-bit *g* field for an operation code, the 3-bit *h* field to designate an address index register, and the 3-bit *i* field to designate a source or result register. (See special register values.)

Figure 2-4 shows the two general applications for the 2-parcel instruction format with combined *j*, *k*, and *m* fields.

NOTE

When using an immediate constant which has both relocatable and parcel attributes, the result of the relocation will be incorrect if the loader-determined actual address (within the user's field length) is greater than 1,048,575. This is because the resulting relocated value will have more than 22 significant bits. A CAL caution message is issued if this occurs. The exception to this is when "A*h exp*" executes on a CRAY X-MP/48.



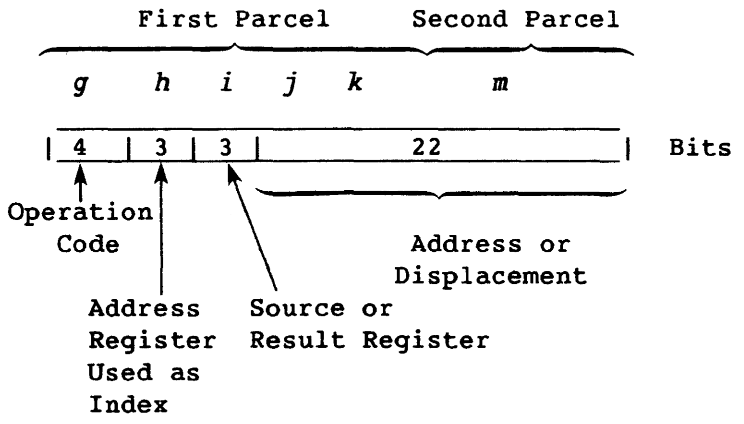


Figure 2-4. 2-parcel Instruction Format with Combined *j, k,* and *m* Fields

2.1.4 2-PARCEL INSTRUCTION FORMAT WITH COMBINED *i, j, k,* AND *m* FIELDS

The 2-parcel branch instruction type uses the combined *i, j, k,* and *m* fields to contain a 24-bit address that allows branching to an instruction parcel (see figure 2-5). A 7-bit operation code (*gh*) is followed by an *ijkm* field. The high-order bit of the *i* field is unused.

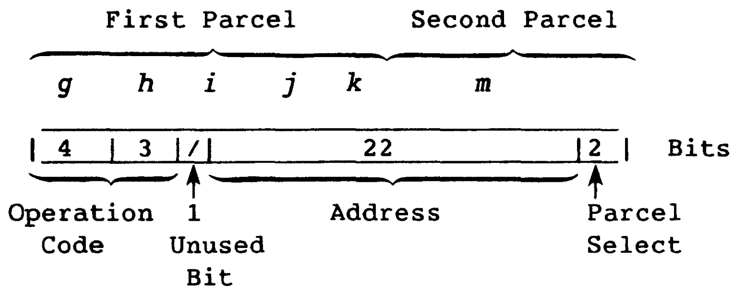


Figure 2-5. 2-parcel Instruction Format with Combined *i, j, k,* and *m* Fields

The 2-parcel instruction type for a 24-bit immediate constant (figure 2-6) uses the combined *i*, *j*, *k*, and *m* fields to hold the constant. This instruction type uses the 4-bit *g* field for an operation code and the 3-bit *h* field to designate the result address register. The high-order bit of the *i* field is set.

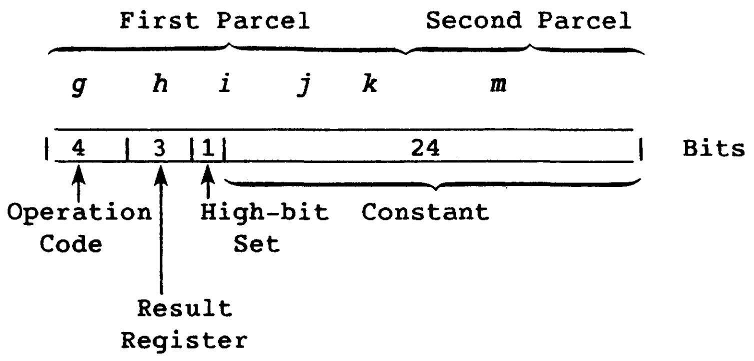


Figure 2-6. 2-parcel Instruction Format for a 24-bit Immediate Constant with Combined *i*, *j*, *k*, and *m* Fields

2.2 SPECIAL REGISTER VALUES

If the S0 and A0 registers are referenced in the *j* or *k* fields of certain instructions, the contents of the respective register is not used; instead, a special operand is generated. The special operand is available regardless of existing A0 or S0 reservations (and in this case is not checked). This use does not alter the actual value of the S0 or A0 register. If S0 or A0 is used in the *i* field as the operand, the actual value of the register is provided. Table 2-1 shows the special register values.

2.3 SYMBOLIC NOTATION

The following information describes the notation used for coding symbolic machine instructions. CAL contains two syntax forms: general and special.

2.3.1 GENERAL SYNTAX

Register designators and the location, result, operand, and comment fields have the following general syntax requirements.

2.3.1.1 Register designators

A, B, SB[†], S, T, ST[†], SM[†], and V registers can be referenced with numeric or symbolic designators. The symbolic designators can be entered uppercase, lowercase, or any mixture of case.

In the symbolic notation, the *h*, *i*, *j*, and *k* designators indicate the field of the machine instruction into which the register designator constant or symbol value is placed. An expression (*exp*) occupies the *jk*, *ijk*, *jkm*, or *ijkm* fields depending on the operation code and magnitude of the expression value.

Supporting registers have the following designators:

Designator	Register
CA	Current Address
CL	Channel Limit
CI	Channel Interrupt Flag
CE	Channel Error Flag
RT	Real-time Clock
MC	Master Clear
SB	Sign Bit (S*k*, with *k*=0)
SM[†]	Semaphore
VL	Vector Length
VM	Vector Mask
XA	Exchange Address

Table 2-1. Special Register Values

Field	Operand Value
A*h*, *h*=0	0
A*i*, *i*=0	(A0)
A*j*, *j*=0	0
A*k*, *k*=0	1
S*i*, *i*=0	(S0)
S*j*, *j*=0	0
S*k*, *k*=0	2**63

† CRAY X-MP Computer Systems only

2.3.1.2 Location field

The location field of a symbolic instruction optionally contains a symbol. When a symbol is present, it is assigned a parcel address as indicated by the current value of the location counter after any required force to parcel boundary occurs.

2.3.1.3 Result field

The result field of a symbolic machine instruction can consist of one, two, or three subfields separated by commas. A subfield can be null or it can contain a register designator or an expression. The expression specifies a memory address which indicates the register or memory location to receive the results of the operation. The result field may contain a mnemonic indicating the function being performed (for example, J for jump or ex for exit). The mnemonics are case sensitive and must be entered in either all uppercase or all lowercase letters, they cannot be mixed. For example, EX is a valid mnemonic for exit, while Ex is not.

2.3.1.4 Operand field

The operand field of a symbolic machine instruction consists of no subfield or one, two, or three subfields separated by commas. A subfield can be null, contain an expression (with no register designators), or consist of register designators and operators.

The following special characters can appear in the operand field of symbolic machine instructions and are used by the assembler in determining the operation to be performed.

Character	Operation
+	Arithmetic sum of specified registers
-	Arithmetic difference of specified registers
*	Arithmetic product of specified registers
/	Reciprocal of approximation
#	Use ones complement
>	Shift value or form mask from left to right
<	Shift value or form mask from right to left
&	Logical product of specified registers
!	Logical sum of specified registers
\	Logical difference of specified registers

In some instructions, register designators are prefixed by the following letters which have special meaning to the assembler. These letters can be entered either uppercase or lowercase (case insensitive).

F	Floating-point operation
H	Half-precision floating-point operation
R	Rounded floating-point operation
I	Reciprocal iteration
P	Population count
Q	Parity count
Z	Leading-zero count

2.3.1.5 Comment field

The comment field of the symbolic machine instructions begins in column 35. By convention, the comment should be preceded by a semicolon (;) in column 35, and a space.

2.3.2 SPECIAL SYNTAX FORMS

The CAL instruction repertoire has been expanded for the convenience of programmers to allow for special forms of symbolic instructions. Because of this expansion, certain Cray machine instructions can be generated from two or more different CAL instructions. For example, both of the following instructions generate instruction 00200, which causes a 1 to be entered into the VL register:

```
VL  A0
VL  1
```

The first instruction is the basic form of the Enter VL instruction, which takes advantage of the special case where (A*k*)=1 if *k*=0; the second instruction is a special syntax form providing the programmer with a more convenient notation for the special case.

Any of the operations performed by special instructions can be performed using instructions in the basic set. Instructions having a special syntax form are identified as such in the instruction description found later in this section.

In several cases, a single syntax form of an instruction can result in any of several different machine instructions being generated. In these cases, which provide for entering the value of an expression into an A register or into an S register or for shifting S register contents, the assembler determines which instruction to generate from characteristics of the expression.

2.4 MONITOR MODE INSTRUCTIONS

The monitor mode instructions (channel control, set real-time clock, and programmable clock interrupts) perform specialized functions that are useful to the operating system. These instructions execute only when the CPU is operating in the monitor mode. If an instruction is executed while not in the monitor mode, it is treated as a no-op.

MACHINE INSTRUCTION DESCRIPTIONS 3

This section contains detailed information about individual instructions or groups of related instructions. Each instruction begins with boxed information consisting of the Cray Assembly Language (CAL) syntax format. This consists of a result field description, an operand field description, a brief description of each instruction, and the machine instruction (octal code sequence defined by the *gh* fields). The appearance of an *m* in a format description designates an instruction consisting of two parcels. An x in the format description signifies that the field containing the x is ignored during CRAY-1 instruction execution. CAL will insert a 0 for each occurrence of x.

Following the boxed information is a detailed description of the instruction or instructions, and an example using the instruction.

****

CAUTION

Instructions with *g, h, i, j, k,* and *m* fields not explicitly described in the following instructions may produce indeterminate results.

****

Specific information about the CPU parameter (including the *primary* and *charac* options) of the CAL invocation statement is found in the following manual:

SR-2003 CAL Assembler Version 2 Reference Manual

Result	Operand	Description	Machine Instruction
ERR		Error exit	000000
ERR†	*exp*	Error exit	000*ijk*

† Special CAL syntax on CRAY-1 Computer Systems only

The 000 instruction is treated as an error condition and an exchange sequence occurs. The contents of the instruction buffers are voided by the exchange sequence. If monitor mode is not in effect, the Error Exit flag in the Flag (F) register is set. All instructions issued before this instruction are run to completion.

When the results of previously issued instructions have arrived at the operating registers, an exchange occurs to the Exchange Package designated by the contents of the Exchange Address (XA) register. The program address stored in the Exchange Package on the terminating exchange sequence is advanced by 1 parcel from the address of the error exit instruction.

The error exit instruction is not generally used in program code. This instruction is used to halt execution of an incorrectly coded program that branches to an unused area of memory or into a data area.

The expression in the operand field is optional and has no effect on instruction execution; the low-order 9 bits of the expression value are placed in the *ijk* fields of the instruction.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
000000		ERR		
000017		ERR	D'15	

INSTRUCTION 0010

Result	Operand	Description	Machine Instruction
CA,A*j*†	A*k*	Set the Current Address (CA) register, for the channel indicated by (A*j*), to (A*k*) and activate the channel	0010*jk*
PASS††		Pass	001000

† Privileged to monitor mode
†† Special CAL syntax

The 0010*jk* instruction sets the Current Address (CA) register for the channel indicated by the contents of A*j* to the value specified in A*k*. It then activates the channel.

Before this instruction is issued, the Channel Limit (CL) register should be initialized. As the transfer progresses, the address in CA is increased. When the contents of CA equals the contents of CL, the transfer is complete for the words at the initial address in CA through 1 less than the address in CL.

When the *j* designator is 0 or when the contents of A*j* is less than 2 or greater than 25, the instruction executes as a pass instruction. When the *k* designator is 0, CA is set to 1.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
001035		CA,A3	A5	
001000		Pass		

INSTRUCTION 0011

Result	Operand	Description	Machine Instruction
CL,A*j*†	A*k*	Set the channel (A*j*) limit address to (A*k*)	0011*jk*

† Privileged to monitor mode

The 0011*jk* instruction sets the Channel Limit (CL) register for the channel indicated by the contents of A*j* to the address specified in A*k*.

The instruction is usually issued before issuing the CA,A*j* A*k* instruction.

When the *j* designator is 0 or when the contents of A*j* is less than 2 or greater than 25, the instruction is executed as a pass instruction. When the *k* designator is 0, CL is set to 1.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
001134		CL,A3	A4	

INSTRUCTION 0012

Result	Operand	Description	Machine Instruction
CI,A*j*†		Clear Channel (A*j*) Interrupt flag	0012*j*0
MC,A*j*††		Clear Channel (A*j*) Interrupt flag and Error flag; set device master-clear (output channel); clear device ready-held (input channel)	0012*j*1

† Privileged to monitor mode

†† Privileged to monitor mode on CRAY X-MP Computer Systems only

Instruction 0012*j*0 clears the Interrupt flag and Error flag for the channel indicated by the contents of A*j*.

When the *j* designator is 0 or when the contents of A*j* is less than 2 or greater than 25, the instruction is executed as a pass instruction.

Instruction 0012*j*1 sets the device Master Clear. If (A*j*) represents an output channel, the master clear is set; if (A*j*) represents an input channel, the ready flag is cleared.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
001210		CI,A1		
001241		MC,A4		
001201		MC,A0		

INSTRUCTION 0013

Result	Operand	Description	Machine Instruction
XA[†]	A*j*	Enter XA register with (A*j*)	0013*j*0

† Privileged to monitor mode

The 0013*j*0 instruction transmits bits 12 through 19 of register A*j* to the Exchange Address (XA) register.

If the *j* designator is 0, 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 (EX).

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
001350		XA	A5	

INSTRUCTION 0014

Result	Operand	Description	Machine Instruction
RT	S*j*	Enter RTC with (S*j*)	0014*j*0
SIPI†	*exp*	Set interprocessor interrupt request of CPU *exp*; 0≤*exp*≤3	0014*j*1
SIPI† ††		Set interprocessor interrupt request	001401
CIPI†		Clear interprocessor interrupt	001402
CLN† †††	*exp*	Cluster number = *exp* where 0≤*exp*≤5	0014*j*3
PCI¶	S*j*	Set program interrupt interval	0014*j*4
CCI¶		Clear clock interrupt	001405
ECI¶		Enable clock interrupts	001406
DCI¶		Disable clock interrupts	001407

† CRAY X-MP Computer Systems with two or four CPUs. This instruction is available when the numeric trait NUMCPUS, which is specified on the CPU parameter of the CAL invocation statement, is greater than one.

†† Special CAL syntax

††† CRAY X-MP Computer Systems only. This instruction is available when the numeric trait NUMCLSTR, which is specified on the CPU parameter of the CAL invocation statement, is greater than zero.

¶ Programmable clock (optional on CRAY-1 Models A and B). This instruction is available through the logical trait PC specified on the CPU parameter of the CAL invocation statement.

NOTE

Instruction 0014 is privileged to monitor mode and is treated as a pass instruction if the monitor mode bit is not set.

The 0014*j*0 instruction transmits the contents of register S*j* to the Real-time Clock register. When the *j* designator is 0, the Real-time Clock register is set to 0.

The 001401 and 001402 instructions handle interprocessor interrupt requests. When the *k* designator is 1, the instruction sets the internal CPU interrupt request in another CPU. If the other CPU is not in monitor mode, the ICP (Interrupt from Internal CPU) flag sets in the F register, causing an interrupt. The request remains until cleared by the receiving CPU.

When the *k* designator is 2, the instruction clears the internal CPU interrupt request set by another CPU.

The 0014*j*3 instruction sets the cluster number to *j* to make the following cluster selections:

CLN = 0	No cluster; all shared register and semaphore operations are no-ops, (except SB, ST, or SM register reads, which return a 0 value to A*i* or S*i*).
CLN = 1	Cluster 1
CLN = 2	Cluster 2
CLN = 3	Cluster 3
CLN = 4	Cluster 4
CLN = 5	Cluster 5

Each of clusters 1, 2, 3, 4, and 5 has a separate set of SM, SB, and ST registers.

The 0014*j*4 instruction loads the low-order 32 bits from the S*j* register into the Interrupt Interval register (II) and the Interrupt Countdown counter (ICD). The Interrupt Countdown counter is a 32-bit counter that is decreased by one each clock period until the contents of the counter is equal to 0. At this time, the real-time clock (RTC) interrupt request is set. The counter is then set to the interval value held in the Interrupt Interval register and repeats the countdown to 0 cycle. When an RTC interrupt request is set, it remains set until a clear clock interrupt (CCI) instruction is executed.

The 001405 instruction clears an RTC interrupt.

The 001406 instruction enables RTC interrupts at a rate determined by the value in the Interrupt Interval (II) register.

The 001407 instruction disables RTC interrupts until an enable clock interrupt (ECI) instruction is executed.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
001420		RT	S2	; Set clock to ; low-order 32 ; bits
001400		RT	S0	; Set clock to 0
001401		SIPI	1	; Set ; interprocessor ; interrupt ; request
001402		CIPI		; Clear ; interprocessor ; interrupt ; request
001403		CLN	0	
001413		CLN	1	
001423		CLN	2	
001433		CLN	3	
001434		PCI	S3	; Load the ; low-order 32 ; bits from (S3) ; to (II)
001405		CCI		; Clear clock ; interrupt
001406		ECI		; Enable clock ; interrupt
001407		DCI		; Disable clock ; interrupt

INSTRUCTION 0015[†]

Result	Operand	Description	Machine Instruction
		Select performance monitor	0015*j*0
		Set maintenance read mode	001501
		Load diagnostic checkbyte with S1	001511
		Set maintenance write mode 1	001521
		Set maintenance write mode 2	001531

NOTE

The 0015 instructions are not supported by CAL at this time.

Instruction 0015*j*0 selects one of four groups of hardware related events to be monitored by the performance counters.

Instructions 001501 through 001531 check the operation of the modules concerned with SECDED and to verify error detection and correction.

Instructions 001501 and 001521 verify check bit memory storage. Instructions 001511 and 001531 verify error detection and correction.

[†] CRAY X-MP Computer Systems only

INSTRUCTION 0020

Result	Operand	Description	Machine Instruction
VL	A*k*	Transmit (A*k*) to VL	00200*k*
VL[†]	1	Enter 1 into VL	002000

† Special CAL syntax

Instruction 00200*k* and its special form (002000) enter the low-order 7 bits of the contents of register A*k* into the VL register.

The contents of the VL register determines the number of operations performed by a vector instruction. Since a vector register has 64 elements, from 1 to 64 operations can be performed. The number of operations is (VL) modulo 64. When (VL) is 0, the number of operations performed is 64.

In this publication, a reference to register V*i* implies operations involving the first *n* elements where *n* is the vector length unless a single element is explicitly noted as in the instructions S*i* V*j*,A*k* and V*i*,A*k* S*j*.

Vector operations controlled by the contents of VL begin with element 0 of the vector registers and operate on consecutive elements.

Examples:

In the first example, if (A3)=6 then (VL)=6 following instruction execution and subsequent vector instructions operate on elements 0 through 5 of vector registers.

Code generated	Location	Result	Operand	Comment
	1	10	20	35
002003		VL	A3	

In the second example, since the *k* designator is assembled as 0, (VL)=1 and vector instructions operate on only one element, element 0.

Code generated	Location	Result	Operand	Comment
	1	10	20	35
002000		VL	1	

Lastly, if (A5)=0, then (VL)=64 and vector instructions operate on all 64 elements of the vectors.

Code generated	Location	Result	Operand	Comment
	1	10	20	35
002005		VL	A5	

INSTRUCTIONS 0021 - 0027

Result	Operand	Description	Machine Instruction
EFI		Enable floating-point interrupt	002100
DFI		Disable floating-point interrupt	002200
ERI†		Enable interrupt on address range error	002300
DRI†		Disable interrupt on address range error	002400
DBM†		Disable bidirectional memory transfers	002500
EBM†		Enable bidirectional memory transfers	002600
CMR†		Complete memory references	002700

† CRAY X-MP Computer Systems only

The EFI and DFI instructions provide for setting and clearing the Floating-point Interrupt flag in the Mode register. These instructions do not check the previous state of the flag.

****

CAUTION

The operating system has status bits reflecting whether interrupts on floating-point range errors are enabled or disabled. Such software status bits need to be modified to agree with the Floating-point Mode flag.

****

The ERI and DRI instructions set and clear the Operand Range Mode flag in the Mode register. The two instructions do not check the previous state of the flag. When set, the Operand Range Mode flag enables interrupts on operand address range errors.

The DBM and EBM instructions disable and enable the bidirectional memory mode. Block reads and writes can operate concurrently in bidirectional memory mode. If the bidirectional memory mode is disabled, only block reads can operate concurrently.

The CMR instruction assures completion of all memory references within a particular CPU issuing the instruction. This instruction does not issue until all memory references before this instruction are at the stage of execution where completion occurs in a fixed amount of time. For example, a load of any data that has been stored by the CPU issuing instruction CMR is assured of receiving the updated data if the load is issued after the CMR instruction. Synchronization of memory references between processors can be done by this instruction in conjunction with semaphore instructions.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
002300		ERI		
002400		DRI		
002500		DBM		
002600		EBM		
002700		CMR		

Result	Operand	Description	Machine Instruction
VM	S*j*	Transmit (S*j*) to VM	0030*j*0
VM[†]	0	Clear VM	003000
SM*jk*[††]	1,TS	Test and set semaphore *jk*, $0 \le jk \le 31$ (decimal)	0034*jk*
SM*jk*[††]	0	Clear semaphore *jk*, $0 \le jk \le 31$ (decimal)	0036*jk*
SM*jk*[††]	1	Set semaphore *jk*, $0 \le jk \le 31$ (decimal)	0037*jk*

† Special CAL syntax
†† CRAY X-MP Computer Systems only

Instruction 0030*j*0 and its special form transmit the contents of register S*j* to the VM register. The VM register is zeroed if the *j* designator is 0; the special form accommodates this case.

This instruction may be used in conjunction with the vector merge instructions where an operation is performed depending on the contents of the VM register.

Instruction 0034*jk* tests and sets the semaphore designated by *jk*. If the semaphore is set, issue is held until another CPU clears that semaphore. If the semaphore is clear, the instruction issues and sets the semaphore.

If all CPUs in a cluster are holding issue on a test and set, the DL flag is set in the Exchange Package (if it is not in monitor mode) and an exchange occurs. If an interrupt occurs while a test and set instruction is holding in the CIP register, the WS flag in the Exchange Package sets, CIP and NIP registers clear, and an exchange occurs with the P register pointing to the test and set instruction.

The SM register is 32 bits with SM0 being the most significant bit.

The 0036*jk* instruction clears the semaphore designated by *jk*.

Instruction 0037*jk* sets the semaphore designated by *jk*.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
003040		VM	S4	
003000		VM	0	; Clear VM
003407		SM7	1,TS	
003607		SM7	0	
003707		SM7	1	

INSTRUCTION 0040

Result	Operand	Description	Machine Instruction
EX		Normal exit	004000
EX†	*exp*	Normal exit	004*ijk*

† Special CAL syntax on CRAY-1 Computer Systems only

Instruction 004000 and its special form cause an exchange sequence. The contents of the instruction buffers are voided by the exchange sequence. If monitor mode is not in effect, the Normal Exit flag in the F register is set. All instructions issued before this instruction are run to completion.

When the results of previously issued instructions have arrived at the operating registers, an exchange occurs to the Exchange Package designated by the contents of the Exchange Address (XA) register. The program address stored in the executing Exchange Package is advanced 1 parcel from the address of the normal exit instruction. This instruction is used to issue a monitor request from a user program, or to transfer control from a monitor program to another program.

The expression in the operand field is optional and has no effect on instruction execution; the low-order 9 bits of the expression value are placed in the *ijk* fields of the instruction.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
004000		EX		
004027		EX	27	

INSTRUCTION 0050

Result	Operand	Description	Machine Instruction
J	B*jk*	Jump to (B*jk*)	0050*jk*

The 0050*jk* unconditional branch instruction sets the P register to the parcel address specified by the contents of register B*jk*. Execution continues at that address.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
005017		J	B17	
005003		J	B.RTNADDR	RTNADDR=03 (octal

INSTRUCTION 0060

Result	Operand	Description	Machine Instruction
J	*exp*	Jump to *exp*	006*ijkm*

The 006*ijkm* unconditional branch instruction sets the P register to the parcel address specified by the low-order 24 bits of the expression. Execution continues at that address.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
006 00002124b+		J	TAG1	
006 00001753a+		J	LDY3+1	
006 00004533c+		J	*+3	

INSTRUCTION 0070

Result	Operand	Description	Machine Instruction
R	*exp*	Return jump to *exp;* set B00 to (P)+2	007*ijkm*

Instruction 007*ijkm* sets register B00 to the address of the parcel following the instruction. The P register is then set to the parcel address specified by the low-order 24 bits of the expression. Execution continues at that address.

The purpose of the instruction is to provide a return linkage for subroutine calls. The subroutine is entered via a return jump. The subroutine returns to the caller at the instruction following the call by executing a branch to the contents of the B register containing the saved address.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
007 00001142d+		R	HELP	

Result	Operand	Description	Machine Instruction
JAZ	*exp*	Branch to *exp* if (A0)=0	010*ijkm*
JAN	*exp*	Branch to *exp* if (A0)≠0	011*ijkm*
JAP	*exp*	Branch to *exp* if (A0) positive	012*ijkm*
JAM	*exp*	Branch to *exp* if (A0) negative	013*ijkm*

NOTE

When executing the above instructions on CRAY X-MP/48, the high-order bit of *i* must be 0.

The above instructions test the contents of A0 for the specified condition. If the condition is satisfied, the P register is set to the parcel address specified by the low-order 24 bits of the expression. Execution continues at that address.

If the condition is not satisfied, execution continues with the instruction following the branch instruction. For the JAP and JAM instructions, a 0 value in A0 is considered positive.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
010 00002243d+		JAZ	TAG3+2	
011 00004520a+		JAN	P.CON1	
012 00002221c+		JAP	TAG2	
013 00002124b+		JAM	TAG1	

Result	Operand	Description	Machine Instruction
JSZ	*exp*	Branch to *exp* if (S0)=0	014*ijkm*
JSN	*exp*	Branch to *exp* if (S0)≠0	015*ijkm*
JSP	*exp*	Branch to *exp* if (S0) positive	016*ijkm*
JSM	*exp*	Branch to *exp* if (S0) negative	017*ijkm*

NOTE

When executing the above instructions on CRAY X-MP/48, the high-order bit of *i* must be 0.

The above instructions test the contents of S0 for the specified condition. If the condition is satisfied, the P register is set to the parcel address specified by the low-order 24 bits of the expression. Execution continues at that address.

If the condition is not satisfied, execution continues with the instruction following the branch instruction. For the JSP and JSM instructions, a zero value in S0 is considered positive.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
014 00002221c+		JSZ	TAG2	
015 00002124d+		JSN	TAG1+2	
016 00004533c+		JSP	*+3	
017 00002367c+		JSM	TAG4	

INSTRUCTION 01*h*

Result	Operand	Description	Machine Instruction
A*h*†	*exp*	Transmit *ijkm* to A*h*; where the high-order bit of *i* is 1	01*hijkm*

† CRAY X-MP Computer Systems only. This instruction is available through the logical trait EMA specified on the CPU parameter of the CAL invocation statement, and CAL will then generate one of these instructions: 01*h*, 022, or 031.

Instruction 01*h* will not be generated if NOEMA is specified.

This instruction enters a 24-bit value into A*h* that is composed of the low-order 24 bits of the *ijkm* field. The high-order bit of the *ijkm* field must be set to distinguish the 01*h* instruction from the 010 to 017 branches.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
0a 0114 00000200		A1	O'200	
c 0174 00001001		A7	SYMBOL	
	SYMBOL	=	O'1001	

Result	Operand	Description	Machine Instruction
A*i*†	*exp*	Enter *exp* into A*i*	020*ijkm* or 021*ijkm* or 022*ijk*

† These instructions are available through the logical trait NOEMA specified on the CPU parameter of the CAL invocation statement, and CAL will generate one of these instructions: 020, 021, 022, 031.

Instructions 020 and 021 wil not be generated of EMA is specified.

The above instruction enters a quantity into A*i*. The syntax differs from most CAL symbolic instructions in that the assembler generates any of three Cray machine instructions depending on the form, value, and attributes of the expression.

The assembler generates an instruction 022*ijk* where the *jk* fields contain the 6-bit value of the expression if all of the following conditions are true:

- The value of the expression is positive and less than 64
- All symbols (if any) are previously defined within the expression
- The expression has a relative attribute of absolute

If any of the conditions are not true, the assembler generates either the 2-parcel instruction 020*ijkm* or 021*ijkm*. If the expression has a positive value, or has a relative attribute of either relocatable or external, instruction 020*ijkm* is generated with the value entered in the 22-bit *jkm* field. If the expression value is negative and has a relative attribute of absolute, instruction 021*ijkm* is generated with the ones complement of the expression value entered into the 22-bit *jkm* field except where the *exp* value is explicitly "-1".

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
022310		A3	O'10	
0212 00000010		A2	#O'10	
	AREG	=	2	
0212 00000007		A.AREG	-O'10	
0202 00000130		A2	O'130	
0203 00000021		A3	VAL+1	; VAL=20 (octal)
0204 01777777		A4	O'1777777	
0205 00051531		A5	A'SY'R	
0226 00000000		A6	#MINUS1	; MINUS1=-1
		EXT	X	
0204 17777777		A4	X-1	; 020*ijkm* used if
				; expression is
				; external

INSTRUCTION 023

Result	Operand	Description	Machine Instruction
A*i*	S*j*	Transmit (S*j*) to A*i*	023*ij*0
A*i*†	VL	Transmit (VL) to A*i*	023*i*01

† CRAY X-MP Computer Systems only

Instruction 023*ij*0 transmits the low-order 24 bits of the contents of register S*j* to register A*i*. A*i* is zeroed if the *j* designator is 0.

Instruction 023*i*01 enters the contents of the VL register into A*i*.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
023420		A4	S2	
023201		A2	VL	

Result	Operand	Description	Machine Instruction
A*i*	B*jk*	Transmit (B*jk*) to A*i*	024*ijk*
B*jk*	A*i*	Transmit (A*i*) to B*jk*	025*ijk*

Instruction 024*ijk* enters the contents of register B*jk* into register A*i*.

Instruction 025*ijk* enters the contents of register A*i* into register B*jk*.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
024517		A5	B17	
	SVNTN	=	O'17	
024517		A5	B.SVNTN	
025634		B34	A6	
025634		B.THRTY4	A6	; THRTY4=34 (octal

INSTRUCTION 026

Result	Operand	Description	Machine Instruction
A*i*	PS*j*	Population count of (S*j*) to A*i*	026*ij*0
A*i*[†]	QS*j*	Population count parity of (S*j*) to A*i*	026*ij*1
A*i*[††]	SB*j*	Transfer (SB*j*) to A*i*	026*ij*7

† Population Count (optional on CRAY-1 Models A and B)
†† CRAY X-MP Computer Systems only

Instruction 026*ij*0 counts the number of 1 bits in the contents of S*j* and enters the result into A*i*. A*i* is zeroed if the *j* designator is 0.

Instruction 026*ij*1 enters a 0 in A*i* if S*j* has an even number of 1 bits in S*j* and enters a 1 in S*j* if it has an odd number of 1 bits.

These two instructions execute in the Scalar Leading Zero/Population Count functional unit.

Instruction 026*ij*7 transfers the contents of the SB*j* register shared between the CPUs to A*i*.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
026720		A7	PS2	; Pop count of ; S2 to A7
026271		A2	QS7	; Pop count ; parity of ; S7 to A2
026007		A0	SB0	
026017		A0	SB1	

INSTRUCTION 027

Result	Operand	Description	Machine Instruction
A*i*	ZS*j*	Leading zero count of (S*j*) to A*i*	027*ij*0
SB*j*†	A*i*	Transfer (A*i*) to SB*j*	027*ij*7

† CRAY X-MP Computer Systems only

Instruction 027*ij*0 counts the number of leading zeros in the contents of S*j* and enters the result into A*i*. A*i* is set to 64 if the *j* designator is 0, or if the S*j* register contains 0.

This instruction executes in the Scalar Leading Zero/Population Count functional unit.

Instruction 027*ij*7 transfers the contents of register A*i* into register SB*j*, which is shared between the CPUs in the current cluster.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
027130		A1	ZS3	
027007		SB0	A0	
027107		SB0	A1	

Result	Operand	Description	Machine Instruction
A*i*	A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*	030*ijk*
A*i*†	A*j*+1	Integer sum of (A*j*) and 1 to A*i*	030*ij*0
A*i*†	A*k*	Transmit (A*k*) to A*i*	030*i*0*k*
A*i*	A*j*-A*k*	Integer difference of (A*j*) less (A*k*) to A*i*	031*ijk*
A*i*†	A*j*-1	Integer difference of (A*j*) less 1 to A*i*	031*ij*0
A*i*†	-A*k*	Transmit negative of (A*k*) to A*i*	031*i*0*k*
A*i*†	-1	Enter -1 into A*i*	031*i*00

† Special CAL syntax

Instruction 030*ijk* and its special form (030*ij*0) add the contents of register A*j* to the contents of register A*k* and enter the result into register A*i*. A*k* is transmitted to A*i* when the *j* designator is 0 and the *k* designator is nonzero. The value 1 is transmitted to A*i* when the *j* and *k* designators are both 0. (A*j*)+1 is transmitted to A*i* when the *j* designator is nonzero and the *k* designator is 0. The assembler allows an alternate form of the instruction when the *k* designator is 0.

The instruction executes in the Address Integer Add functional unit.

Instruction 030*i*0*k* enters the contents of register A*k* into register A*i*. The value 1 is entered if the *k* designator is 0.

The instruction 030*i*0*k* executes in the Address Integer Add functional unit.

Instruction 031*ijk* and its special form (031*ij*0) subtract the contents of register A*k* from the contents of register A*j* and enter the result into register A*i*. The negative of A*k* is transmitted to A*i* when the *j* designator is 0 and the *k* designator is nonzero. A -1 is transmitted to A*i* when the *j* and *k* designators are both 0. (A*j*)-1 is transmitted to A*i* when the *j* designator is nonzero and the *k* designator is 0.

The instruction 031*ijk* executes in the Address Integer Add functional unit.

The special form represents the case where (A*k*)=1 if *k*=0.

Instruction 031*i*0*k* enters the negative (twos complement) of the contents of register A*k* into register A*i*. The value -1 is entered into A*i* if the *k* designator is 0.

The instruction 031*i*0*k* executes in the Address Integer Add functional unit.

Instruction 031*i*00 is generated in place of instruction 020*ijkm* if the operand is explicitly -1.

This instruction executes in the Address Add functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
030123		A1	A2+A3	
030102		A1	A2	
030230		A2	A3+1	
030602		A6	A2	
031456		A4	A5-A6	
031102		A1	-A2	
031450		A4	A5-A1	
031703		A7	-A3	
031300		A3	-1	

INSTRUCTION 032

Result	Operand	Description	Machine Instruction
A*i*	A*j**A*k*	Integer product of (A*j*) and (A*k*) to A*i*	032*ijk*

Instruction 032*ijk* forms the integer product of the contents of register A*j* and register A*k* and enters the low-order 24 bits of the result into A*i*. A*i* is cleared when the *j* designator is 0. A*j* is transmitted to A*i* when the *k* designator is 0 and the *j* designator is nonzero.

The instruction executes in the Address Integer Multiply functional unit. There is no overflow detection.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
032712		A7	A1*A2	

INSTRUCTION 033

Result	Operand	Description	Machine Instruction
A*i*	CI	Channel number of highest priority interrupt request to A*i*	033*i*00
A*i*	CA,A*j*	Address of channel (A*j*) to A*i* (*j*≠0)	033*ij*0
A*i*	CE,A*j*	Error flag of channel (A*j*) to A*i*	033*ij*1

Instruction 033*i*00 enters the channel number of the highest priority interrupt request into A*i*.

Instruction 033*ij*0 enters the contents of the Current Address (CA) register for the channel specified by the contents of A*j* into register A*i*.

Instruction 033*ij*1 enters the error flag for the channel specified by the contents of A*j* into the low-order 7 bits of A*i*. The high-order bits of A*i* are cleared. The error flag can be cleared only in monitor mode using the CI,A*j* instruction, or the CRAY X-MP instruction MC,A*j*.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
033100		A1	CI	
033230		A2	CA,A3	
033341		A3	CE,A4	

Result	Operand	Description	Machine Instruction
B*jk*,A*i*	,A0	Read (A*i*) words starting at B*jk* from memory starting at (A0)	034*ijk*
B*jk*,A*i*†	0,A0	Read (A*i*) words starting at B*jk* from memory starting at (A0)	034*ijk*
,A0	B*jk*,A*i*	Store (A*i*) words starting at B*jk* to memory starting at (A0)	035*ijk*
0,A0†	B*jk*,A*i*	Store (A*i*) words starting at B*jk* to memory starting at (A0)	035*ijk*
T*jk*,A*i*	,A0	Read (A*i*) words starting at T*jk* from memory starting at (A0)	036*ijk*
T*jk*,A*i*†	0,A0	Read (A*i*) words starting at T*jk* from memory starting at (A0)	036*ijk*
,A0	T*jk*,A*i*	Store (A*i*) words starting at T*jk* to memory starting at (A0)	037*ijk*
0,A0†	T*jk*,A*i*	Store (A*i*) words starting at T*jk* to memory starting at (A0)	037*ijk*

† Special CAL syntax

Instruction 034*ijk* and its special form are used to transfer words from memory directly into B registers. A0 contains the address of the first word of memory to be transferred. The *jk* designator specifies the first B register to be used in the transfer. The low-order 24 bits of consecutive words of memory are loaded into consecutive B registers.

Processing of B registers is circular. B00 is loaded after B77 if the count specified in A*i* is not exhausted after B77 is loaded. The low-order 7 bits of the contents of A*i* specify the number of words transmitted. Wraparound occurs if the low-order 7-bits of (A*i*) are greater than 64.

If (A*i*)=0, no words are transferred. Note also that if *i*=0, (A0) is used for the block length as well as the starting memory address. The CAL assembler issues a warning message in this case.

Instruction 035*ijk* and its special form are used to store words from B registers directly into memory. A0 contains the address of the first word of memory to receive data. The *jk* designator specifies the first B register to be used in the transfer. Subsequent B register contents are stored in consecutive words of memory.

Processing of B registers is circular. B00 is processed after B77 if the count specified in A*i* is not exhausted after B77 is processed. The low-order 7 bits of the contents of A*i* specify the number of words transmitted. Wraparound occurs if the low-order 7-bits of A*i* are greater than 64.

If (A*i*)=0, no words are transferred. Note also that if *i*=0, (A0) is used for the block length as well as the starting memory address. The CAL assembler issues a warning message in this case.

Instruction 036*ijk* and its special form are used to transfer words from memory directly into T registers. A0 contains the address of the first word of memory to be transferred. The *jk* designator specifies the first T register to be used in the transfer. The loading of T registers is circular. T00 is loaded after T77 if the count specified in A*i* is not exhausted after T77 is loaded. The low-order 7 bits of the contents of A*i* specify the number of words transmitted. Wraparound occurs if the low-order 7-bits of A*i* are greater than 64.

If (A*i*)=0, no words are transferred. If *i*=0, (A0) is used for the block length and the starting memory address. The CAL assembler issues a warning message in this case.

Instruction 037*ijk* and its special form are used to store words from T registers directly into memory. A0 contains the address of the first word of memory to receive data. The *jk* designator specifies the first T register to be used in the transfer. Subsequent T register contents are stored in consecutive words of memory. Processing of T registers is circular. T00 is processed after T77 if the count specified in A*i* is not exhausted after T77 is processed. The low-order 7 bits of the contents of register A*i* specify the number of words transmitted. Wraparound occurs if the low-order 7-bits of A*i* are greater than 64.

If (A*i*)=0, no words are transferred. Note also that if *i*=0, (A0) is used for the block length as well as the starting memory address, and CAL issues a warning message.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
034407		B7,A4	,A0	
	BB	=	O'22	
	FWAR	=	5	
034522		B.BB,A.FWAR	0,A0	
035522		,A0	B22,A5	
	BB	=	O'22	
	FWAR	=	5	
035522		0,A0	B.BB,A.FWAR	
036407		T7,A4	,A0	
	TT	=	O'22	
	FWAR	=	5	
036522		T.TT,A.FWAR	0,A0	
37522		,A0	T22,A5	
	TT	=	O'22	
	FWAR	=	5	
037522		0,A0	T.TT,A.FWAR	

Result	Operand	Description	Machine Instruction
S*i*	*exp*	Enter *exp* into S*i*	040***ijkm*** or 041***ijkm***

The above instruction enters a quantity into S*i*. Either the 2-parcel 040***ijkm*** instruction or the 2-parcel 041***ijkm*** instruction is generated, depending on the value of the expression.

If the expression has a positive value or a relative attribute of either relocatable or external, instruction 040***ijkm*** is generated with the 22-bit ***jkm*** field containing the expression value. If the expression has a negative value and a relative attribute of absolute, instruction 041***ijkm*** is generated with the 22-bit ***jkm*** field containing the ones complement of the expression value.

Refer to the 042-043 instructions for additional information on S*i* *exp* instructions.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
0402 00000130		S2	O'130	
	SREG	=	3	
0403 00000021		S.SREG	VAL+1	; VAL=20 (octal)
0404 01777777		S4	O'1777777	
0405 00051531		S5	A'SY'R	
			3	
0406 00000000		S6	#MINUS1	; MINUS1=-1
0413 00000002		S3	#2	
0414 01777776		S4	-O'1777777	
0414 00000003		S4	#VAL2	; VAL2=3
		EXT	X	
0401 17777777		S1	X-1	; 040***ijkm*** used
				; if expression
				; is external

Result	Operand	Description	Machine Instruction
S*i*	<*exp*	Form ones mask in S*i* from right	042*ijk*
S*i*†	#>*exp*	Form zeros mask in S*i* from left	042*ijk*
S*i*†	1	Enter 1 into S*i*	042*i*77
S*i*†	-1	Enter -1 into Si	042*i*00
S*i*†	0	Clear S*i*	043*i*00
S*i*	>*exp*	Form ones mask in S*i* from left	043*ijk*
S*i*†	#<*exp*	Form zeros mask in S*i* from right	043*ijk*

† Special CAL syntax

Instruction 042*ijk* generates a mask of ones from the right. The assembler evaluates the expression to determine the mask length.

In the first instruction, the mask length is the value of the expression. In the second instruction, the mask length is 64 minus the expression value. The mask length must be a positive integer not exceeding 64; 64 minus the mask length is inserted into the *jk* fields of the instruction. If the value of the expression is 0 for the first instruction or 64 for the second instruction, the assembler generates instruction 043*i*00.

Instruction 042*ijk* executes in the Scalar Logical functional unit.

Instructions 042*i*77, 042*i*00, and 043*i*00 are initially recognized by the assembler as the symbolic instruction S*i* *exp*. The assembler then checks the expression to see if it has one of these three forms. If it finds one of the forms in the exact syntax shown, it generates the corresponding Cray machine instruction. If none of these forms is found, instruction 040*ijkm* or 041*ijkm* is generated. These special forms allow more efficient instructions for entering often used values into S1.

Instructions 043*i*00, 042*i*77, and 042*i*00 execute in the Scalar Logical functional unit.

Instruction 043*ijk* generates a mask of ones from the left. The assembler evaluates the expression to determine the mask length.

In instruction 043*ijk,* the mask length is the value of the expression. In the special syntax form, the mask length is 64 minus the expression value. The mask length must be a positive integer not exceeding 64 and is inserted into the *jk* fields of the instruction. If the expression value is 64 for the first instruction or 0 for the second instruction, the assembler generates instruction 042*i*00.

Instruction 043*ijk* executes in the Scalar Logical functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
042200		S2	-1	
042273		S2	<5	
042273		S2	#>O'73	
042366		S3	<D'10	
042400		S4	<O'100	
043500		S5	<0	
043600		S6	0	; Clear S6
042677		S6	1	; Set S6 to 1
043205		S2	>5	
043205		S2	#<O'73	
043500		S5	<0	

Result	Operand	Description	Machine Instruction
S*i*	S*j*&S*k*	Logical product of (S*j*) and (S*k*) to S*i*	044*ijk*
S*i*†	S*j*&SB	Sign bit of (S*j*) to S*i*	044*ij*0
S*i*†	SB&S*j*	Sign bit of (S*j*) to S*i*; *j*≠0	044*ij*0
S*i*	#S*k*&S*j*	Logical product of (S*j*) and #(S*k*) to S*i*	045*ijk*
S*i*†	#SB&S*j*	(S*j*) with sign bit cleared to S*i*	045*ij*0
S*i*	S*j*\S*k*	Logical difference of (S*j*) and (S*k*) to S*i*	046*ijk*
S*i*†	S*j*\SB	Enter (S*j*) into S*i* with sign bit toggled	046*ij*0
S*i*†	SB\S*j*	Enter (S*j*) into S*i* with sign bit toggled; *j*≠0	046*ij*0
S*i*	#S*j*\S*k*	Logical equivalence of (S*j*) and (S*k*) to S*i*	047*ijk*
S*i*†	#S*j*\SB	Logical equivalence of (S*j*) and sign bit to S*i*	047*ij*0
S*i*†	#SB\S*j*	Logical equivalence of sign bit and (S*j*) to S*i*; *j*≠0	047*ij*0

† Special CAL syntax

NOTE

When the above instructions execute on a CRAY X-MP, SB with no register designator is the sign bit, not Shared Address register.

Result	Operand	Description	Machine Instruction
S*i*†	#S*k*	Transmit ones complement of (S*k*) to S*i*	047*i*0*k*
S*i*†	#SB	Enter ones complement of sign bit in S*i*	047*i*00
S*i*	S*j*!S*i*&S*k*	Scalar merge of (S*i*) and (S*j*) to S*i*	050*ijk*
S*i*†	S*j*!S*i*&SB	Scalar merge of (S*i*) and sign bit of (S*j*) to S*i*	050*ij*0
S*i*	S*j*!S*k*	Logical sum of (S*j*) and (S*k*) to S*i*	051*ijk*
S*i*†	S*j*!SB	Logical sum of (S*j*) and sign bit to S*i*	051*ij*0
S*i*†	SB!S*j*	Logical sum of sign bit and (S*j*) to S*i*; *j*≠0	051*ij*0
S*i*†	S*k*	Transmit (S*k*) to S*i*	051*i*0*k*
S*i*†	SB	Enter sign bit into S*i*	051*i*00

† Special CAL syntax

NOTE

When the above instructions execute on a CRAY X-MP, SB with no register designator is the sign bit, not Shared Address register.

Instruction 044*ijk* forms the logical product of the contents of S*j* and S*k* and enters the result into S*i*. If the *j* and *k* designators have the same nonzero value, the contents of S*j* is transmitted to S*i*.

If the *j* designator is 0, register S*i* is zeroed. If the *j* designator is nonzero and the *k* designator is 0, the sign bit of the contents of S*j* is extracted. The two special forms of the instruction accommodate this case. The two forms perform identical functions, but *j* must not be equal to 0 in the second form. If *j* is equal to 0, an assembly error results.

Instruction 045*ijk* forms the logical product of the contents of S*j* and the ones complement of the contents of S*k* and enters the result into S*i*. If the *j* and *k* designators have the same value or if the *j* designator is 0, register S*i* is zeroed.

If the *j* designator is nonzero and the *k* designator is 0, the contents of S*j* with the sign bit cleared is transmitted to S*i*. The special syntax form accommodates this case.

Instruction 046*ijk* forms the logical difference of the contents of S*j* and the contents of S*k* and enters the result into S*i*. If the *j* and *k* designators have the same nonzero value, S*i* is zeroed.

If the *j* designator is 0 and the *k* designator is nonzero, the contents of S*k* is transmitted to S*i*. If the *j* designator is nonzero and the *k* designator is 0, the sign bit of the contents of S*j* is complemented and the result is transmitted to S*i*. The two special syntax forms provide for this case. The two forms perform identical functions; however, in the second form, *j* must not equal 0. If *j* equals 0, an assembly error results.

Instruction 047*ijk* forms the logical equivalence of the contents of S*j* and the contents of S*k* and enters the result into S*i*. Bits of S*i* are set to 1 when the corresponding bits of the contents of S*j* and the contents of S*k* are both 1 or both 0.

If the *j* and *k* designators have the same nonzero value, the contents of S*i* is set to all ones. If the *j* designator is 0 and the *k* designator is nonzero, the ones complement of the contents of S*k* is transmitted to S*i*. If the *j* designator is nonzero and the *k* designator is 0, all bits other than the sign bit of the contents of S*j* are complemented and the result is transmitted to S*i*.

The two special forms of the instruction accommodate this case. The two forms perform identical functions; however, in the second form, *j* must not equal 0. If *j* equals 0, an error results.

Instruction 047*i*0*k* forms the ones complement of the contents of register S*k* and enters the value into S*i*. The complement of the sign bit is entered into S*i* if the *k* designator is 0.

Instruction 047*i*00 clears the sign bit and sets all other bits.

Instructions 050*ijk* and 050*ij*0 merge the contents of S*j* with the contents of S*i* depending on the ones mask in S*k*.

The result is defined by (S*j*&S*k*)!(S*i*&#S*k*) 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. S*i* bits are cleared when the corresponding S*k* bits are 1 if the *j* designator is 0 and the *k* designator is nonzero. The sign bit of S*j* replaces the sign bit of S*i* if the *j* designator is nonzero and the *k* designator is 0 as provided for by the special syntax form of the instruction. The sign bit of S*i* is cleared if the *j* and *k* designators are both 0.

Instruction 051*ijk* forms the logical sum of the contents of S*j* and the contents of S*k* and enters the result into S*i*. If the *j* and *k* designators have the same nonzero value, the contents of S*j* are transmitted to S*i*. If the *j* designator is 0 and the *k* designator is nonzero, the contents of S*k* are transmitted to S*i*.

If the *j* designator is nonzero and the *k* designator is 0, the contents of S*j* with the sign bit set to 1 are transmitted to S*i*. The two special syntax forms provide for this case. If the *j* and *k* designators are both 0, a ones mask consisting of only the sign bit is entered into S*i*.

The two special forms perform an identical function but in the second form $j \neq 0$; if $j=0$, an assembly error results.

Instruction 051*i*0*k* enters the contents of register S*k* into register S*i*. The sign bit is set to 1 in S*i* if the *k* designator is 0.

Instruction 051*i*00 can be used to set the sign bit of S*i* and zero all other bits.

Instructions 044*ijk* through 051 execute in the Scalar Logical functional unit.

Example:

Code generated	Location 1	Result 10	Operand 20	Comment 35
044235		S2	S3&S5	
044655		S6	S5&S5	; S5 to S6
044160		S1	S6&SB	; Get sign of S6
044160		S1	SB&S6	; Get sign of S6
045271		S2	#S1&S7	
045430		S4	#SB&S3	; Clear sign bit ; of S3 and ; enter into S4
045506		S5	#S6&S0	; Clear S5
045670		S6	#SB&S7	; Clear sign bit
046123		S1	S2\S3	
046455		S4	S5\S5	; Clear S4
046506		S5	S0\S6	; S6 to S5
046770		S7	S7\SB	; Toggle sign ; bit
047345		S3	#S4\S5	
047260		S2	#S6\SB	
047260		S2	#SB\S6	
047203		S2	#S3	
047200		S2	#SB	
050123		S1	S2!S1&S3	
050760		S7	S6!S7&S0	

Example (continued):

Code generated	Location	Result	Operand	Comment
	1	10	20	35
051472		S4	S7!S2	
051366		S3	S6!S6	
051710		S7	SB!S1	
051701		S7	S1	
	I	=	1	
051100		S.I	SB	

Result	Operand	Description	Machine Instruction
S0	S*i*<*exp*	Shift (S*i*) left *exp* places to S0	052*ijk*
S0	S*i*>*exp*	Shift (S*i*) right *exp* places to S0	053*ijk*
S*i*	S*i*<*exp*	Shift (S*i*) left *exp* places to S*i*	054*ijk*
S*i*	S*i*>*exp*	Shift (S*i*) right *exp* places to S*i*	055*ijk*

Instruction 052*ijk* shifts the contents of S*i* to the left by the amount specified by the expression and enters the result into S0. The shift count must be a positive integer value not exceeding 64. The shift is end off with zero fill. If the shift count is 64, instruction 053000 is generated and S0 is zeroed.

Instruction 053*ijk* shifts the contents of S*i* to the right by the amount specified by the expression and enters the result into S0. The shift count must be a positive integer value not exceeding 64. The assembler stores 64 minus the shift count in the *jk* field of the instruction. The shift is end off with zero fill. If the shift count is 0, instruction 052000 is generated and the contents of S0 is not altered.

Instruction 054*ijk* shifts the contents of S*i* to the left by the amount specified by the expression and enters the result into S*i*. The shift count must be a positive integer value not exceeding 64. The shift is end off with zero fill. If the shift count is 64, instruction 055*i*00 is generated and S*i* is zeroed.

Instruction 055*ijk* shifts the contents of S*i* to the right by the amount specified by the expression and enters the result into S*i*. The shift count must be a positive integer value not exceeding 64. The assembler stores 64 minus the shift count in the *jk* field of the instruction. If the shift count is 0, instruction 054*i*00 is generated and the contents of S*i* is not altered. The shift is end off with zero fill.

Instructions 052*ijk*, 053*ijk*, 054*ijk*, and 055*ijk* execute in the Scalar Shift functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
052305		S0	S3<5	
052724		S0	S7<VAL+4	
053373		S0	S3>5	
053066		S0	S0>D'10	
053754		S0	S7>VAL+4	
052100		S0	S1>0	
054703		S7	S7<3	
054622		S6	S6<VAL+2	
055775		S7	S7>3	
055656		S6	S6>VAL+2	

Result	Operand	Description	Machine Instruction
S*i*	S*i*,S*j*<A*k*	Left shift by (A*k*) of (S*i*) and (S*j*) to S*i*	056*ijk*
S*i*†	S*i*,S*j*<1	Left shift by 1 of (S*i*) and (S*j*) to S*i*	056*ij*0
S*i*†	S*i*<A*k*	Left shift by (A*k*) of (S*i*) to S*i*	056*i*0*k*
S*i*	S*j*,S*i*>A*k*	Right shift by (A*k*) of (S*j*) and (S*i*) to S*i*	057*ijk*
S*i*†	S*j*,S*i*>1	Right shift by 1 of (S*j*) and (S*i*)to S*i*	057*ij*0
S*i*†	S*i*>A*k*	Right shift by (A*k*) of (S*i*) to S*i*	057*i*0*k*

† Special CAL syntax

Instruction 056*ijk* and its special forms produce a 128-bit quantity by concatenating the contents of S*i* and the contents of S*j*, shifting the resulting value to the left by an amount specified by the low-order bits of A*k* and entering the high-order bits of the result into S*i*. The shift is end off with zero fill.

Replacing the A*k* reference with 1 is the same as setting the *k* designator to 0; a reference to A0 provides a shift count of 1. Omitting the S*j* reference is the same as setting the *j* designator to 0; the contents of S*i* are concatenated with a word of zeros.

S*i* is cleared if the shift count exceeds 127. The shift is a left circular shift of the contents of S*i* if the shift count does not exceed 64 and the *i* and *j* designators are equal and nonzero. The instruction produces the same result as the S*i* S*i*<*exp* instruction if the shift count does not exceed 63 and the *k* designator is 0. The contents of S*j* are not affected if the *i* and *j* designators are unequal.

Instruction 057*ijk* and its special forms produce a 128-bit quantity by concatenating the contents of S*j* and the contents of S*i*, shifting the resulting value to the right by an amount specified by the low-order 7 bits of the contents of A*k* and entering the low-order bits of the result into S*i*. The shift is end off with zero fill.

Replacing the A*k* reference with 1 is the same as setting the *k* designator to 0; a reference to A0 provides a shift count of 1. Omitting the S*j* reference is the same as setting the *j* designator to 0; the contents of S*i* are concatenated with a word of zeros.

S*i* is cleared if the shift count exceeds 127. The shift is a right circular shift of the contents of S*i* if the shift count does not exceed 64 and the *i* and *j* designators are equal and nonzero. The instruction produces the same result as the S*i* S*i*>*exp* instruction if the shift count does not exceed 63 and the *j* designator is 0. The contents of S*j* are not affected if the *i* and *j* designators are unequal.

Instruction 056*ijk* and 057*ijk* executes in the Scalar Shift functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
056235		S2	S2,S3<A5	
056340		S3	S3,S4<1	; Left 1 place
056604		S6	S6<A4	
057235		S2	S3,S2>A5	
057604		S6	S6>A4	
057340		S3	S4,S3>1	; Right 1 place

Result	Operand	Description	Machine Instruction
S*i*	S*j*+S*k*	Integer sum of (S*j*) and (S*k*) to S*i*	060*ijk*
S*i*	S*j*-S*k*	Integer difference of (S*j*) less (S*k*) to S*i*	061*ijk*
S*i*†	-S*k*	Transmit negative of (S*k*) to S*i*	061*i*0*k*

† Special CAL syntax

Instruction 060*ijk* adds the contents of register S*k* to the contents of register S*j* and enters the result into S*i*. S*k* is transmitted to S*i* if the *j* designator is 0 and the *k* designator is nonzero. The sign bit is entered in S*i* and all other bits of S*i* are cleared if the *j* and *k* designators are both 0.

Instruction 061*ijk* subtracts the contents of register S*k* from the contents of register S*j* and enters the result into S*i*. The high-order bit of S*i* is set and all other bits of S*i* are cleared when the *j* and *k* designators are both 0. The negative (twos complement) of S*k* is transmitted to S*i* if the *j* designator is 0 and the *k* designator is nonzero.

Instruction 061*i*0*k* enters the negative (twos complement) of the contents of S*k* into S*i*. The sign bit is set if the *k* designator is 0.

Instructions 060*ijk*, 061*ijk*, 061*i*0*k* execute in the Scalar Integer Add functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
060237		S2	S3+S7	
060405		S4	S0+S5	
061123		S1	S2-S3	
061506		S5	-S6	

Result	Operand	Description	Machine Instruction
S*i*	S*j*+FS*k*	Floating-point sum of (S*j*) and (S*k*) to S*i*	062*ijk*
S*i*†	+FS*k*	Normalize (S*k*) to S*i*	062*i*0*k*
S*i*	S*j*-FS*k*	Floating-point difference of (S*j*) less (S*k*) to S*i*	063*ijk*
S*i*†	-FS*k*	Transmit the negative of (S*k*) as a normalized floating-point value	063*i*0*k*

† Special CAL syntax

Instruction 062*ijk* and its special form produce the floating-point sum of the contents of the S*j* and S*k* registers and enters the result into S*i*. The result is normalized even if the operands are unnormalized. The *k* designator is not normally 0. In the special form, the *j* designator is assumed to be 0 so that the normalized contents of S*k* are entered into S*i*.

Instruction 063*ijk* forms the floating-point difference of the contents of register S*j* less the contents of register S*k*, and enters the normalized result into S*i*. The result is normalized even if the operands are unnormalized.

The negative (twos complement) of the floating-point quantity in S*k* is transmitted to S*i* as a normalized floating-point number if the *j* designator is 0 and the *k* designator is nonzero. The special form accommodates this special case. The *k* designator is normally nonzero.

Instructions 062*ijk*, 063*ijk*, and 063*i*0*k* execute in the Floating-point Add functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
062345		S3	S4+FS5	
062404		S4	+FS4	
063302		S3	-FS2	
063761		S7	S6-FS1	

Result	Operand	Description	Machine Instruction
S*i*	S*j**FS*k*	Floating-point product of (S*j*) and (S*k*) to S*i*	064*ijk*
S*i*	S*j**HS*k*	Half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*	065*ijk*
S*i*	S*j**RS*k*	Rounded floating-point product of (S*j*) and (S*k*) to S*i*	066*ijk*
S*i*	S*j**IS*k*	2-floating-point product of (S*j*) and (S*k*) to S*i*	067*ijk*

Instruction 064*ijk* forms the floating-point product of the contents of S*j* and S*k* and enters the result into S*i*. The result is not normalized if either operand is unnormalized.

Instruction 065*ijk* forms the half-precision rounded floating-point product of the contents of the S*j* and S*k* registers and enters the result into S*i*. The result is not normalized if either operand is unnormalized. The low-order 18 bits of the result are zeroed. This instruction can be used in a divide algorithm when only 30 bits of accuracy are required.

Instruction 066*ijk* forms the rounded floating-point product of the contents of the S*j* and S*k* registers and enters the result into S*i*. The result is not normalized if either operand is unnormalized. This operation is used in the reciprocal approximation sequence.

Instruction 067*ijk* forms 2 minus the floating-point product of the contents of S*j* and S*k* and enters the result into S*i*. The result is not normalized if either operand is unnormalized.

Instructions 064*ijk*, 065*ijk*, 066*ijk*, and 067*ijk* execute in the Floating-point Multiply functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
064234		S2	S3*FS4	
065167		S1	S6*HS7	
066147		S1	S4*RS7	
067324		S3	S2*IS4	

Result	Operand	Description	Machine Instruction
S*i*	/HS*j*	Floating-point reciprocal approximation of (S*j*)to S*i*	070*ij*0

Instruction 070*ij*0 forms an approximation to the reciprocal of the floating-point value in S*j* and enters the result into S*i*. The result is meaningless if the contents of S*j* is unnormalized or 0. This instruction is used in the divide sequence as illustrated in the following example.

Instruction 070*ij*0 executes in the Floating-point Reciprocal functional unit.

Example:

```
|Code generated     |Location|Result    |Operand     |Comment
|__________________ |1       |10        |20          |35
|                   |        |          |            |
|                   |*       |Divide S1 by S2; result to S1
|070320             |        |S3        |/HS2        |; Approximate
|                   |        |          |            |; reciprocal
|                   |        |          |            |
|064113             |        |S1        |S1*FS3      |; Approximate
|                   |        |          |            |; result
|                   |        |          |            |
|067223             |        |S2        |S2*IS3      |; Correction
|                   |        |          |            |; factor
|                   |        |          |            |
|064112             |        |S1        |S1*FS2      |
|                   |        |          |            |
|                   |        |          |            |
|                   |*       |Divide S1 by S2 with result accurate to
|                   |*       |30 bits   |            |
|070320             |        |S3        |/HS2        |
|                   |        |          |            |
|065313             |        |S3        |S1*HS3      |
```

Example (continued):

Code generated	Location	Result	Operand	Comment
	1	10	20	35
	*	Integer divide A1 by A2; Result to A3		
071222		S2	+FA2	; Denominator
071121		S1	+FA1	; Numerator
062202		S2	S0+FS2	; Normalize
062101		S1	S0+FS1	
070220		S2	/HS2	; Reciprocal ; approximation ; to 1/D
065110		S1	S1*HS2	; Rounded ; half-precision ; multiply
071230		S2	0.6	
062112		S1	S1+FS2	; Fix quotient
023310		A3	S1	; 24-bit signed ; result to A3

INSTRUCTION 071

Result	Operand	Description	Machine Instruction
S*i*	A*k*	Transmit (A*k*) to S*i* without sign extension	071*i*0*k*
S*i*	+A*k*	Transmit (A*k*) to S*i* with sign extension	071*i*1*k*
S*i*	+FA*k*	Transmit (A*k*) to S*i* as an unnormalized floating-point value	071*i*2*k*
S*i*	0.6	Enter 0.75*(2**48) into S*i* as normalized floating-point constant	071*i*30
S*i*	0.4	Enter 0.5 into S*i* as normalized floating-point constant	071*i*40
S*i*	1.	Enter 1 into S*i* as normalized floating-point constant	071*i*50
S*i*	2.	Enter 2 into S*i* as normalized floating-point constant	071*i*60
S*i*	4.	Enter 4 into S*i* as normalized floating-point constant	071*i*70

Instruction 071*i*0*k* transfers the 24-bit value in register A*k* into the low-order 24 bits of register S*i*. The value is treated as an unsigned integer. The high-order bits of S*i* are zeroed. A value of 1 is entered into S*i* when the *k* designator is 0.

Instruction 071*i*1*k* transfers the 24-bit value in register A*k* into the low-order 24 bits of register S*i*. The value is treated as a signed integer and the sign bit of the contents of register A*k* is extended to the high-order bits of S*i*. A value of 1 is entered into S*i* when the *k* designator is 0.

Instruction 071*i*2*k* transmits the contents of register A*k* to S*i* as an unnormalized floating-point value. The result can then be added to 0 to normalize. When the *k* designator is 0, an unnormalized floating-point 1 is entered into S*i*.

Instructions 071*i*3*k* through 071*i*7*k* are initially recognized by the assembler as the symbolic instruction S*i* *exp*. The assembler then checks the expression to see if it has any of the indicated forms. If it finds one of the instructions in the exact syntax shown, it generates the corresponding Cray machine instruction. If none of the indicated forms are found, instruction 040*ijkm* or 041*ijkm* is generated as previously described. These special forms allow more efficient instructions for entering commonly used values into S*i*.

The syntax form S*i* 0.6 (071*i*30) is useful for extracting the integer part of a floating-point quantity (that is, fix) as illustrated in the examples.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
071707		S7	A7	
071717		S7	+A7	
071324		S3	+FA4	
	FIX	=	6	
071630		S.FIX	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	; Floating-point ; integer part
063113		S1	S1-FS3	; Fractional ; part

Result	Operand	Description	Machine Instruction
S*i*	RT	Transmit (RTC) to S*i*	072*i*00
S*i*†	SM	Read semaphore to S*i*	072*i*02
S*i*†	ST*j*	Read (ST*j*) register to S*i*	072*ij*3
S*i*††	VM	Transmit (VM) to S*i*	073*i*00
		Read performance counter into S*i*	073*i*11
		Increased performance counter	073*i*21
		Clear all maintenance modes	073*i*31
S*i*†††	SR*j*	Transmit (SR*j*) to S*i*; *j*=0	073*ij*1
SM†	S*i*	Load semaphores from S*i*	073*i*02
ST*j*†	S*i*	Transfer (S*i*) to ST*j*	073*ij*3
S*i*	T*jk*	Transmit (T*jk*) to S*i*	074*ijk*
T*jk*	S*i*	Transmit (S*i*) to T*jk*	075*ijk*

† CRAY X-MP Computer Systems only. This instruction is available when the numeric trait NUMCLSTR, which is specified on the CPU parameter of the CAL invocation statement, is greater than zero.

†† Not supported by CAL at this time

††† CRAY X-MP computer systems only. This instruction is available through the logical trait STATRG specified on the CPU parameter of the CAL invocation statement.

Instruction 072*i*00 enters the 64-bit contents of the real-time clock into register S*i*. The clock is increased by 1 each clock period. The real-time clock can be reset only when in monitor mode using instruction 072*i*00.

Instruction 072*i*02 enters the values of all of the semaphores into S*i*. The 32-bit SM register is left justified in S*i* with SM00 occupying the sign bit.

Instruction 072*ij*3 enters the contents of register ST*j* into register S*i*.

Instruction 073*i*00 enters the 64-bit contents of the VM register into register S*i*. The VM register is normally read after having been set by instruction 1750*jk*.

Instruction 073*ij*1 enters the contents of the Status register into S*i*.

Instruction 073*i*02 sets the semaphores from 32 high-order bits of S*i*. SM00 receives the sign bit of S*i*.

Instruction 073*ij*3 transfers the contents of register S*i* into register ST*j*, which is shared between the CPUs in the current cluster.

Instruction 074*ijk* enters the contents of register T*jk* into register S*i*.

Instruction 075*ijk* enters the contents of register S*i* into register T*jk*.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
072700		S7	RT	
072002		S0	SM	
072602		S6	SM	
072003		S0	ST0	
072013		S0	ST1	
073200		S2	VM	
073001		S0	SR0	
073301		S3	SR0	
073002		SM	S0	
073102		SM	S1	
073502		SM	S5	
073003		ST0	S0	
073103		ST0	S1	

Example: (continued)

Code generated	Location	Result	Operand	Comment
	1	10	20	35
074306		S3	T6	
074566		S5	T66	
075306		T6	S3	
075567		T67	S5	

Result	Operand	Description	Machine Instruction
S*i*	V*j*,A*k*	Transmit (V*j*, element (A*k*) to S*i*	076*ijk*
V*i*,A*k*	S*j*	Transmit (S*j*) to V*i* element (A*k*)	077*ijk*
V*i*,A*k*†	0	Clear element (A*k*) of register V*i*	077*i*0*k*

† Special CAL syntax

Instruction 076*ijk* enters the contents of the element of V*j* indicated by the contents of the low-order 6 bits of A*k* into S*i*. The second element (that is, element 1) is selected if the *k* designator is 0.

Instruction 077*ijk* transmits the contents of register S*j* to an element of V*i* as determined by the low-order 6 bits of the contents of A*k*. Element 1, the second element of V*i*, is selected if the *k* designator is 0.

Instruction 077*i*0*k* zeros element (A*k*) of register V*i*. The low-order 6 bits of A*k* determine which element is zeroed. The second element of register V*i* is zeroed (that is, element 1) if the *k* designator is 0.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
076456		S4	V5,A6	
	I	=	4	
	J	=	5	
	K	=	6	
076456		S.I	V.J,A.K	
077167		V1,A7	S6	
077602		V6,A2	0	

INSTRUCTIONS 10*h* - 13*h*

Result	Operand	Description	Machine Instruction
A*i*	*exp*,A*h*	Read from ((Ah) + *exp*) to A*i*	10*hijkm*
A*i*†	*exp*,0	Read from (*exp*) to A*i*	100*ijkm*
A*i*†	*exp*,	Read from (*exp*) to A*i*	100*ijkm*
A*i*†	,A*h*	Read from (A*h*) to A*i*	10*hi*000
exp,A*h*	A*i*	Store (A*i*) to (A*h*) + *exp*	11*hijkm*
exp,0†	A*i*	Store (A*i*) to *exp*	110*ijkm*
exp,†	A*i*	Store (A*i*) to *exp*	110*ijkm*
,A*h*†	A*i*	Store (A*i*) to (A*h*)	11*hi*000
S*i*	*exp*,A*h*	Read from ((A*i*) + *exp*) to S*i*	12*hijkm*
S*i*†	*exp*,0	Read from (*exp*) to S*i*	120*ijkm*
S*i*†	*exp*,	Read from (*exp*) to S*i*	120*ijkm*
S*i*†	,A*h*	Read from (A*h*) to S*i*	12*hi*000
exp,A*h*	S*i*	Store (S*i*) to (A*h*) + *exp*	13*hijkm*
exp,0†	S*i*	Store (S*i*) to *exp*	130*ijkm*
exp,†	S*i*	Store (S*i*) to *exp*	130*ijkm*
,A*h*†	S*i*	Store (S*i*) to (A*h*)	13*hi*000

† Special CAL syntax

For these instructions, only the value of the expression is used if the *h* designator is 0 or if a zero or blank field is used in place of A*h*. Only the content of A*h* is used if the expression is omitted. An expression, if present, must not have a parcel-address attribute or an assembly error occurs.

Instructions 10*hijkm* through 10*hi*000 load the low-order 24 bits of a memory word directly into an A register. The memory address is determined by adding the address in the register A*h* to the expression value. Only the value of the expression is used if the *h* designator is 0, or a 0 or blank field is used in place of A*h*. Only the contents of A*h* is used if the expression is omitted. An assembler error will occur if an expression has a parcel-address attribute

Instructions 11*hijkm* through 11*hi*000 store 24 bits from register A*i* directly into memory. The high-order bits of the memory word are zeroed. The memory address is determined by adding the address in register A*h* to the expression value.

Instructions 12*hijkm* through 12*hi*000 load the contents of a memory word directly into an S register. The memory address is determined by adding the address in register A*h* to the expression value. Only the value of the expression is used if the *h* designator is 0 or a zero or blank field is used in place of A*h*. Only the contents of A*h* is used if the expression is omitted. An assembler error will occur if an expression has a parcel-address attribute.

Instructions 13*hijkm* through 13*hi*000 store the contents of register S*i* directly into memory. The memory address is determined by adding the address in register A*h* to the expression value.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
1001 00004520+		A1	CON1,A0	
1002 00004520+		A2	CON1,0	
1013 00004521+		A3	CON1+1,A1	
1024 17777777+		A4	-1,A2	
1005 00003000+		A5	ADDR,	
1046 00004647+		A6	CON,A4	
1046 00000000+		A6	,A4	
1061 00000001+		A1	1,A6	
1072 00000177+		A2	O'177,A7	
1101 00004520+		CON1,A0	A1	
1102 00004520+		CON1,0	A2	
1113 00004521+		CON1+1,A1	A3	
1124 17777777+		-1,A2	A4	
1105 00003000+		ADDR,	A5	
1146 00004647+		CON,A4	A6	
1146 00000000+		,A4	A6	
1161 00000001+		1,A6	A1	
1172 00000177+		O'177,A7	A2	

Example: (continued)

Code generated	Location	Result	Operand	Comment
	1	10	20	35
1201 00004520+		S1	CON1,A0	
1202 00004520+		S2	CON1,0	
1213 00004521+		S3	CON1+1,A1	
1224 17777777+		S4	-1,A2	
1205 00003000+		S5	ADDR,	
1246 00004647+		S6	CON,A4	
1246 00000000+		S6	,A4	
1261 00000001+		S1	1,A6	
1272 00000177+		S2	O'177,A7	
1301 00004520+		CON1,A0	S1	
1302 00004520+		CON1,0	S2	
1346 00000000+		,A4	S6	
1324 17777777+		-1,A2	S4	
1305 00003000+		ADDR,	S5	

Result	Operand	Description	Machine Instruction
V*i*	S*j*&V*k*	Logical products of (S*j*) and (V*k*) to V*i*	140*ijk*
V*i*	V*j*&V*k*	Logical products of (V*j*) and (V*k*) to V*i*	141*ijk*
V*i*	S*j*!V*k*	Logical sums of (S*j*) and (V*k*) to V*i*	142*ijk*
V*i*†	V*k*	Transmit (V*k*) to V*i*	142*i*0*k*
V*i*	V*j*!V*k*	Logical sums of (V*j*) and (V*k*) to V*i*	143*ijk*
V*i*	S*j*\V*k*	Logical differences of (S*j*) and (V*k*) to V*i*	144*ijk*
V*i*	V*j*\V*k*	Logical differences of (V*j*) and (V*k*) to V*i*	145*ijk*
V*i*†	0	Clear V*i*	145*iii*
V*i*	S*j*!V*k*&VM	Vector merge of (S*j*) and (V*k*) to V*i*	146*ijk*
V*i*†	#VM&V*k*	Vector merge of (V*k*) and zero to V*i*	146*i*0*k*
V*i*	V*j*!V*k*&VM	Vector merge of (V*j*) and (V*k*) to V*i*	147*ijk*

† Special CAL syntax

Instruction 140*ijk* forms the logical products of the contents of S*j* and the contents of elements of V*k* and enters the results into elements of V*i*. If the *j* designator is 0, elements of register V*i* are zeroed. The number of operations performed by this instruction is determined by the contents of the VL register.

Instruction 141*ijk* forms the logical products of the contents of elements of register V*j* and elements of register V*k* and enters the results into elements of V*i*. If the *j* designator is the same as the *k* designator, the contents of the V*j* elements are transmitted to the V*i* elements.

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

Instruction 142*ijk* forms the logical sums of the contents of S*j* and the contents of elements of V*k* and enters the results into elements of V*i*. The contents of the V*k* elements are transmitted to the V*i* elements if the *j* designator is 0. The VL register determines the number of operations performed by this instruction.

Instruction 142*i*0*k* transmits the contents of the elements of register V*k* to the elements of register V*i*. The VL register determines the number of elements performed by this instruction.

Instruction 143*ijk* forms the logical sums of the contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*.

If the *j* and *k* designators are equal, the contents of the V*j* elements are transmitted to V*i*. The VL register determines the number of operations performed by this instruction.

Instruction 144*ijk* forms the logical differences of the contents of S*j* and the contents of elements of V*k* and enters the results into elements of V*i*. If the *j* designator is 0, the contents of the V*k* elements are entered into the V*i* elements. The VL register determines the number of operations performed by this instruction.

Instruction 145*ijk* forms the logical differences of the contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*. If the *j* and *k* designators are equal, the V*i* elements are zeroed. The VL register determines the number of operations performed by this instruction.

Instruction 145*iii* zeros elements of V*i*. The VL register determines the number of elements performed by this instruction.

Instruction 146*ijk* transmits the contents of S*j* or the contents of element *n* of V*k* to element *n* of V*i* depending on the ones mask in the VM register. The content of S*j* is transmitted if bit *n* of VM is 1; the content of element *n* of V*k* is transmitted if bit *n* of VM is 0.

Element *n* of V*i* is 0 if the *j* designator is 0 and bit *n* of VM is 1. The VL register determines the number of operations performed by this instruction.

Instruction 146*i*0*k* zeroes element *n* of register V*i* or transmits the contents of element *n* of V*k* to element *n* of V*i* depending on the ones mask in the VM register. If bit *n* of VM is 1, element *n* of V*i* is zeroed; if bit *n* is 0, element *n* of V*k* is transmitted. The VL register determines the number of operations performed by this instrction.

Instruction 147*ijk* transmits the contents of element *n* of V*j* or element *n* of V*k* to element *n* of V*i* depending on the ones mask in the VM register. The content of the V*j* element is transmitted if bit *n* of VM is 1; the content of the V*k* element is transmitted if bit *n* of VM is 0. The VL register determines the number of operations performed by this instruction.

Instructions 140*ijk* through 147*ijk* execute in the Vector Logical functional unit.

For these instructions (except 145*iii*), a warning level message is issued if the logical trait VRECUR is specified on the CPU parameter of the CAL invocation statement and either *i*=*j* or *i*=*k* (for V registers only). A comment level message is issued of NOVRECUR is specified on the CPU parameter of the CAL invocation statement.

Examples:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
140123		V1	S2&V3	
141257		V2	V5&V7	
141033		V0	V3&V3	
142615		V6	S1!V5	
142102		V1	V2	
143714		V7	V1!V4	
144267		V2	S6\V7	
145513		V5	V1\V3	
145500		V5	0	

Examples: (continued)

Code generated	Location	Result	Operand	Comment
	1	10	20	35
146726		V7	S2!V6&VM	

For the above instruction, assume the following initial register conditions exist:

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

After instruction execution, the first four elements of V7 are 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.

Examples: (continued)

Code generated	Location	Result	Operand	Comment
	1	10	20	35
146607		V6	#VM&V7	

Assume the following initial register conditions for the above instruction:

```
               (VL) = 4
               (VM) = 0 50000 0000 0000 0000 0000
    Element 0 of V7 = 1
    Element 1 of V7 = 2
    Element 2 of V7 = 3
    Element 3 of V7 = 4
```

After instruction execution, the first four elements of V6 have been modified as follows:

```
    Element 0 of V6 = 1
    Element 1 of V6 = 0
    Element 2 of V6 = 3
    Element 3 of V6 = 0
```

Examples: (continued)

Code generated	Location	Result	Operand	Comment
	1	10	20	35
147123		V1	V2!V3&VM	

Assume the following initial register conditions exist for the above instruction:

```
                (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 instruction execution, the first four elements of V*i* 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.

Result	Operand	Description	Machine Instruction
V*i*	V*j*<A*k*	Shift (V*j*) left (A*k*) places to V*i*	150*ijk*
V*i*†	V*j*<1	Shift (V*j*) left one place to V*i*	150*ij*0
V*i*	V*j*>A*k*	Shift (V*j*) right (A*k*) places to V*i*	151*ijk*
V*i*†	V*j*>1	Shift (V*j*) right one place to V*i*	151*ij*0

† Special CAL syntax

Instruction 150*ijk* and its special form shift the contents of the elements of register V*j* to the left by the amount specified by the contents of A*k* and enter the results into the elements of V*i*. The VL register determines the number of elements performed by this instruction. For each element, the shift is end off with zero fill. Elements of V*i* are zeroed if the shift count exceeds 63. Element contents are shifted left one place if the *k* designator is 0; this can be specified through the special form of the instruction.

Instruction 151*ijk* and its special form shift the contents of the elements of register V*j* to the right by the amount specified by the contents of A*k* and enter the results into the elements of V*i*. The VL register determines the number of elements performed by this instruction. For each element, the shift is end off with zero fill. Elements of V*i* are zeroed if the shift count exceeds 63. Element contents are shifted right one place if the *k* designator is 0; a special form of the instruction accommodates this feature.

Instructions 150*ijk* and 151*ijk* execute in the Vector Shift functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
150123		V1	V2<A3	
150450		V4	V5<1	; Left 1 place

Examples: (continued)

Code generated	Location	Result	Operand	Comment
	1	10	20	35
151341		V3	V4>A1	
151450		V4	V5>1	; Right 1 place

Result	Operand	Description	Machine Instruction
V*i*	V*j*,V*j*<A*k*	Double shift (V*j*) left (A*k*) places to V*i*	152*ijk*
V*i*†	V*j*,V*j*<1	Double shift (V*j*) left one place to V*i*	152*ij*0
V*i*	V*j*,V*j*>A*k*	Double shift (V*j*) right (A*k*) places to V*i*	153*ijk*
V*i*†	V*j*,V*j*>1	Double shift (V*j*) right one place to V*i*	153*ij*0

† Special CAL syntax

Instruction 152*ijk* and its special form shift 128-bit quantities from elements of V*j* by the amount specified in A*k* and enter the result into elements of V*i*. Element *n* of V*j* is concatenated with element *n*+1 and the 128-bit quantity is shifted left by the amount specified in A*k*. The shift is end off with zero fill. The high-order 64 bits of the results are transmitted to element *n* of V*i*.

The VL register determines the number of elements performed by this instruction. The last element of V*j*, as determined by VL, is concatenated with 64 bits of zeros. The 128-bit quantities are shifted left one place if the *k* designator is 0; the special form of the instruction accommodates this feature.

Instruction 153*ijk* and its special form shift 128-bit quantities from elements of V*j* by the amount specified in A*k* and enter the result into elements of V*i*. Element *n*-1 of V*j* is concatenated with element *n* and the 128-bit quantity is shifted right by the amount specified in A*k*. The shift is end off with zero fill. The low-order 64 bits are transmitted to element *n* of V*i*.

The VL register determines the number of elements performed by this instruction. The first element of V*j* is concatenated with 64 bits of zeros. The 128-bit quantities are shifted right one place if the *k* designator is 0; the special form of the instruction accommodates this feature.

Instructions 152*ijk* and 153*ijk* execute in the Vector Shift functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
152541		V5	V4,V4<A1	

Assume the following initial register conditions for the above instruction:

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

After instruction execution, the first four elements of V5 have been modified as follows:

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

The remaining elements of V5 are unaltered.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
153026		V0	V2,V2>A6	

Assume the following initial register conditions for the above instruction.

```
                  (VL) = 4
                  (A6) = 3
    Element 0 of V2 = 0 00000 0000 0000 0000 0017
    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 instruction execution, the first four elements of V0 have been modified as follows:

```
    Element 0 of V0 = 0 00000 0000 0000 0000 0001
    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.

Result	Operand	Description	Machine Instruction
V*i*	S*j*+V*k*	Integer sums of (S*j*) and (V*k*) to V*i*	154*ijk*
V*i*	V*j*+V*k*	Integer sums of (V*j*) and (V*k*) to V*i*	155*ijk*
V*i*	S*j*-V*k*	Integer differences of (S*j*) and (V*k*) to V*i*	156*ijk*
V*i*†	-V*k*	Transmit twos complement of (V*k*) to V*i*	156*i*0*k*
V*i*	V*j*-V*k*	Integer differences of (V*j*) less (V*k*) to V*i*	157*ijk*

† Special CAL syntax

Instruction 154*ijk* adds the contents of S*j* to each element of V*k* and enters the results into elements of V*i*. Elements of V*k* are transmitted to V*i* if the *j* designator is 0.

The VL register determines the number of operations performed by this instruction.

Instruction 155*ijk* adds the contents of elements of register V*j* to the contents of corresponding elements of register V*k* and enters the results into elements of register V*i*.

The VL register determines the number of operations performed by this instruction.

Instruction 156*ijk* subtracts the contents of each element of V*k* from the contents of register S*j* and enters the results into elements of register V*i*. The negative (twos complement) of each element of V*k* is transmitted to V*i* if the *j* designator is 0.

The VL register determines the number of operations performed by this instruction.

Instruction 156*i*0*k* transmits the twos complement of the contents of elements of register V*k* to the elements of register V*i*. The VL register determines the number of elements performed by this instruction.

Instruction 157*ijk* subtracts the contents of elements of register V*k* from the contents of corresponding elements of register V*j* and enters the results into elements of register V*i*.

The VL register determines the number of operations performed by this instruction.

Instructions 154*ijk* through 157*ijk* execute in the Vector Integer Add functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
154213		V2	S1+V3	
155456		V4	V5+V6	
156712		V7	S1-V2	
156102		V1	-V2	
157345		V3	V4-V5	

Result	Operand	Description	Machine Instruction
V*i*	S*j**FV*k*	Floating-point products of (S*j*) and (V*k*) to V*i*	160*ijk*
V*i*	V*j**FV*k*	Floating-point products of (V*j*) and (V*k*) to V*i*	161*ijk*
V*i*	S*j**HV*k*	Half-precision rounded floating-point products of (S*j*) and (V*k*) to V*i*	162*ijk*
V*i*	V*j**HV*k*	Half-precision rounded floating-point products of (V*j*) and (V*k*) to V*i*	163*ijk*
V*i*	S*j**RV*k*	Rounded floating-point products of (S*j*) and (V*k*) to V*i*	164*ijk*
V*i*	V*j**RV*k*	Rounded floating-point products of (V*j*) and (V*k*) to V*i*	165*ijk*
V*i*	S*j**IV*k*	2-floating-point products of (S*j*) and (V*k*) to V*i*	166*ijk*
V*i*	V*j**IV*k*	2-floating-point products of (V*j*) and (V*k*) to V*i*	167*ijk*

Instruction 160*ijk* forms the floating-point products of the contents of S*j* and elements of V*k* and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The number of operations performed is determined by the contents of the VL register.

Instruction 161*ijk* forms the floating-point products of the contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The number of operations performed is determined by the contents of the VL register.

Instruction 162*ijk* forms the half-precision rounded floating-point products of the contents of the S*j* register and the contents of elements of the V*k* register and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The low-order 18 bits of the results are zeroed.

The number of operations performed by this instruction is determined by the contents of the VL register. This instruction can be used in a divide algorithm when only 30 bits of accuracy are required.

Instruction 163*ijk* forms the half-precision rounded floating-point products of the contents of elements of the V*j* register and elements of the V*k* register and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The low-order 18 bits of the results are zeroed.

The VL register determines the number of operations performed by this instruction. This instruction can be used in a divide algorithm when only 30 bits of accuracy are required.

Instruction 164*ijk* forms the rounded floating-point products of the contents of the S*j* register and the contents of elements of V*k* and enters the results into elements of V*i*. The results will not be normalized if either operand is unnormalized. The VL register determines the number of operations performed by this instruction.

Instruction 165*ijk* forms the rounded floating-point products of the contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*. The results will not be normalized if either operand is unnormalized. The VL register determines the number of operations performed by this instruction.

Instruction 166*ijk* forms 2 minus the floating-point products of the contents of S*j* and the contents of elements of V*k* and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The VL register determines the number of operations performed by this instruction.

Instruction 167*ijk* forms 2 minus the floating-point products of contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. This instruction is used in the divide sequence. The VL register determines the number of operations performed by this instruction.

Instructions 160*ijk* through 167*ijk* execute in the Floating-point Multiply functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
160627		V6	S2*FV7	
161123		V1	V2*FV3	
162456		V4	S5*HV6	
163712		V7	V1*HV2	
164314		V3	S1*RV4	
165567		V5	V6*RV7	
166123		V1	S2*IV3	
167456		V4	V5*IV6	

Result	Operand	Description	Machine Instruction
V*i*	S*j*+FV*k*	Floating-point sums of (S*j*) and (V*k*) to V*i*	170*ijk*
V*i*†	+FV*k*	Normalize (V*k*) to V*i*	170*i*0*k*
V*i*	V*j*+FV*k*	Floating-point sums of (V*j*) and (V*k*) to V*i*	171*ijk*
V*i*	S*j*-FV*k*	Floating-point differences of (S*j*) less (V*k*) to V*i*	172*ijk*
V*i*†	-FV*k*	Transmit normalized negative of (V*k*) to V*i*	172*i*0*k*
V*i*	V*j*-FV*k*	Floating-point differences of (V*j*) less (V*k*) to V*i*	173*ijk*

† Special CAL syntax

Instruction 170*ijk* forms the floating-point sums of the contents of S*j* and elements of register V*k* to elements of register V*i*. The results are normalized even if the operands are unnormalized. The VL register determines the number of operations performed by this instruction.

The special form of the instruction (170*i*0*k*) normalizes the contents of the elements of V*k* and enters the results into elements of V*i*.

Instruction 171*ijk* forms the floating-point sums of the contents of elements of V*j* and elements of V*k* and enters the results into the elements of register V*i*. The results are normalized even if the operands are unnormalized. The number of operations performed is determined by the contents of the VL register.

Instruction 172*ijk* forms the floating-point differences of the contents of S*j* and elements of register V*k* and enters the results into register V*i*. The results are normalized even if the operands are unnormalized. The negatives (twos complements) of floating-point quantities in elements of V*k* are transmitted to V*i* if the *j* designator is 0. The special form (172*i*0*k*) accommodates this special case. The number of operations performed is determined by the contents of the VL register.

Instruction 173*ijk* forms the floating-point differences of the contents of elements of register V*j* less the contents of elements of registers V*k* and enters the results into elements of register V*i*. The results are normalized even if the operands are unnormalized. The VL register determines the number of operations performed by this instruction.

Instructions 170*ijk* through 173*ijk* execute in the Floating-point Add functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
170712		V7	S1+FV2	
170501		V5	+FV1	; Normalize (V1)
				; to V5
171234		V2	V3+FV4	
172516		V5	S1-FV6	
173712		V7	V1-FV2	

Result	Operand	Description	Machine Instruction
V*i*	/HV*j*	Floating-point reciprocal approximation of (V*j*) to V*i*	174*ij*0

Instruction 174*ij*0 forms the approximations to the reciprocals of the floating-point values in elements of V*j* and enters the results into elements of V*i*. The results are meaningless if the contents of elements are unnormalized or 0. This instruction is used in the divide sequence. The VL register determines the number of operations performed by this instruction.

Instruction 174*ij*0 executes in the Floating-point Reciprocal functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	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 elements of V1 by elements of V2;		
	*	Results accurate to 30 bits, result to V6		
174320		V3	/HV2	
165613		V6	V1*HV3	
	*	Divide S1 by elements of V2; Result to V6		
174320		V3	/HV2	
160413		V4	S1*FV3	
167532		V5	V3*IV2	
161645		V6	V4*FV5	

Result	Operand	Description	Machine Instruction
V*i*†	PV*j*	Population count of (V*j*) to (V*i*)	174*ij*1
V*i*†	QV*j*	Population count parity of (V*j*) to (V*i*)	174*ij*2

† Vector Population Count (optional on CRAY-1 Models A and B)

Instruction 174*ij*1 counts the number of 1 bits in the elements of register V*j* and enters the result into the elements of register V*i*. The VL register determines the number of elements performed by this instruction.

Instruction 174*ij*2 enters a 0 or 1 into the elements of V*i* depending on whether the elements of V*j* have an even or odd number of 1 bits. A 0 is entered into element *n* of V*i* if there is an even number of 1 bits in element *n* of V*j*; a 1 is entered into element *n* of V*i* if there is an odd number of 1 bits in element *n* of V*j*. The number of elements involved is determined by the VL register.

Instructions 174*ij*1 and 174*ij*2 execute in the Reciprocal Approximation functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
174311		V3	PV1	; Pop count of ; V1 to V3
174522		V5	QV2	; Pop count ; parity of V2 ; to V5

INSTRUCTION 175

Result	Operand	Description	Machine Instruction
VM	V*j*,Z	Set VM bits for zero elements of V*j*	1750*j*0
VM	V*j*,N	Set VM bits for nonzero elements of V*j*	1750*j*1
VM	V*j*,P	Set VM bits for positive elements of V*j*	1750*j*2
VM	V*j*,M	Set VM bits for negative elements of V*j*	1750*j*3
V*i*,VM†	V*j*,Z	Set VM bits and register V*i* to V*j*, for zero elements of V*j*	175*ij*4
V*i*,VM†	V*j*,N	Set VM bits and register V*i* to V*j*, for nonzero elements of V*j*	175*ij*5
V*i*,VM†	V*j*,P	Set VM bits and register V*i* to V*j*, for positive elements of V*j*	175*ij*6
V*i*,VM†	V*j*,M	Set VM bits and register V*i* to V*j*, for negative elements of V*j*	175*ij*7

† CRAY X-MP Computer Systems only

Instructions 1750*j*0 through 1750*j*3 create a mask in the VM register. The 64 bits of the VM register correspond to the 64 elements of V*j*. Elements of V*j* are tested for the specified condition. If the condition is true for an element, the corresponding bit is set to 1 in the VM register. If the condition is not true, the bit is zeroed.

The number of elements tested is determined by the contents of the VL register; however, the entire VM register is zeroed before elements of V*j* are tested. If the contents of an element is 0, it is considered positive. Element 0 corresponds to bit 0, element 1 to bit 1, and so on, from left-to-right in the register.

Instructions 175*ij*4 through 175*ij*7 create an identical vector mask as in the above instructions, and in addition create a compressed index list in register V*i* based on the results of testing the contents of the elements of register V*j*.

These instructions execute in the Vector Logical functional unit.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
175050		VM	V5,Z	
175061		VM	V6,N	
175072		VM	V7,P	
175013		VM	V1,M	

Result	Operand	Description	Machine Instruction
V*i*	,A0,A*k*	Read from memory starting at (A0) increased by (A*k*) and load into V*i*	176*i*0*k*
V*i*†	,A0,1	Read from consecutive memory addresses starting with (A0) and load into V*i*	176*i*00
V*i*††	,A0,V*k*	Read from memory using memory address (A0) + (V*k*) and load into V*i*	176*i*1*k*
,A0,A*k*	V*j*	Store (V*j*) to memory starting at (A0) increased by (A*k*)	1770*jk*
,A0,1	V*j*	Store (V*j*) to memory in consecutive addresses starting with (A0)	1770*j*0
,A0,V*k*††	V*j*	Store (V*j*) to memory using memory address (A0) + (V*k*)	1771*jk*

† Special CAL syntax
†† CRAY X-MP Computer Systems only

Instruction 176*i*0*k* and 176*i*00 load words into elements of register V*i* directly from memory. A0 contains the starting memory address. This address is increased by the contents of register A*k* for each word transmitted. The contents of A*k* can be positive or negative allowing both forward and backward streams of references. If the *k* designator is 0 or if 1 replaces A*k* in the operand field of the instruction, the address is increased by 1.

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

For instruction 176*i*1*k*, register elements begin with 0 and are increased by 1 for each transfer. The low-order 24 bits of each element of V*k* contain a signed 24-bit integer which is added to (A0) to obtain the current memory address.

The VL register determines the number of words transferred.

Instructions 1770*jk* and 1770*j*0 store words from elements of register V*j* directly into memory. A0 contains the starting memory address. This address is increased by the contents of register A*k* for each word transmitted. The contents of A*k* can be positive or negative allowing both forward and backward streams of references. If the *k* designator is 0 or if 1 replaces A*k* in the result field of the instruction, the address is increased by 1.

The VL register determines the number of elements transferred.

For instruction 1771*jk*, register elements begin with 0 and are increased by 1 for each transfer. The low-order 24 bits of each element of V*k* contains a signed 24-bit integer which is added to (A0) to obtain the current memory address.

The VL register determines the number of elements transferred.

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
176201		V2	,A0,A1	
176500		V5	,A0,1	
177032		,A0,A2	V3	
177030		,A0,1	V3	

APPENDIX SECTION

SYMBOLIC INSTRUCTION SUMMARY A

This appendix contains two (CRAY X-MP and CRAY-1) symbolic instructions summary charts. It also lists the functional units for both the CRAY X-MP and CRAY-1 Computer Systems.

A.1 FUNCTIONAL UNITS

Instructions other than simple transmits or control operations are performed by specialized hardware known as functional units. Each unit implements an algorithm or a portion of the instruction set. For more information on Functional Units, refer to the appropriate hardware reference manual.

	Clock Periods		
Functional Unit	CRAY-1	CRAY X-MP	Instructions
Address Integer Add	2	2	030, 031
Address Integer Multiply	6	4	032
Scalar Integer Add	3	3	060, 061
Scalar Logical	1	1	042-051
Scalar Shift	2	2	052-055
	3	3	056, 057
Scalar Pop/Parity/	4†	4	026
Leading Zero	3	3	027
Vector Integer Add	3	3	154-157
Vector Logical	2	2	140-147, 175
Second Vector Logical	-	4	140-145
Vector Shift	4	3	150, 151, 153
	-	4	152
Vector Pop/Parity	6†	5	174*ij*1, 174*ij*2
Floating-point Add	6	6	062, 063, 170-173
Floating-point Multiply	7	7	064-067, 160-167
Floating-point Reciprocal	14	14	070, 174*ij*0
Memory (Scalar)	11††	14†††	100-130
	-	17¶	100-130
Memory (Vector)	7¶¶	-	176, 177

† Only with vector population
†† For Serial 1: scalar 10, vector 6
††† 2-and 4-processor X-MP
¶ Single-processor X-MP
¶¶ For CRAY-1 M Series: 8, 9, or 10

A.2 CRAY-1 SYMBOLIC MACHINE INSTRUCTIONS

LOGICAL OPERATIONS

Si	Sj&Sk	Vi	Sj&Vk	Vi	Vj&Vk
Si	Sj&SB				
Si	SB&Sj				
Si	#Sk&Sj				
Si	#SB&Sj				
Si	Sj!Sk	Vi	Sj!Vk	Vi	Vj!Vk
Si	Sj!SB				
Si	SB!Sj				
Si	Sj\Sk	Vi	Sj\Vk	Vi	Vj\Vk
Si	Sj\SB				
Si	SB\Sj				
Si	#Sj\Sk				
Si	#Sj\SB				
Si	#SB\Sj				
		VM	Vj,Z		
		VM	Vj,N		
		VM	Vj,P		
		VM	Vj,M		
Si	Sj!Si&Sk				
Si	Sj!Si&SB	Vi	Sj!Vk&VM	Vi	Vj!Vk&VM
		Vi	#VM&Vk		

FLOATING-POINT OPERATIONS

EFI					
DFI					
Si	Sj+FSk	Vi	Sj+FVk	Vi	Vj+FVk
Si	+FSk	Vi	+FVk		
Si	Sj-FSk	Vi	Sj-FVk	Vi	Vj-FVk
Si	-FSk	Vi	-FVk		
Si	Sj*FSk	Vi	Sj*FVk	Vi	Vj*FVk
Si	Sj*HSk	Vi	Sj*HVk	Vi	Vj*HVk
Si	Sj*RSk	Vi	Sj*RVk	Vi	Vj*RVk
Si	Sj*ISk	Vi	Sj*IVk	Vi	Vj*IVk
Si	/HSj			Vi	/HVj

SHIFT INSTRUCTIONS

```
S0   Si<exp         S0   Si>exp
Si   Si<exp         Si   Si>exp

Si   Si,Sj<Ak       Si   Sj,Si>Ak
Si   Si,Sj<1        Si   Sj,Si>1
Si   Si<Ak          Si   Si>Ak

Vi   Vj<Ak          Vi   Vj>Ak
Vi   Vj<1           Vi   Vj>1

Vi   Vj,Vj<Ak       Vi   Vj,Vj>Ak
Vi   Vj,Vj<1        Vi   Vj,Vj>1
```

REGISTER ENTRY INSTRUCTIONS

```
Ai   exp       Si   <exp
Ai   -1        Si   #>exp

               Si   >exp
Si   exp       Si   #<exp

Si   0         Si   SB
Si   1         Si   #SB
Si   -1
Si   1         Vi,Ak  0
Si   2         Vi   0
Si   4         Si   0.4
Si   0.6
```

PROGRAM BRANCHES AND EXITS

```
J   exp
J   Bjk

JAZ  exp            JSZ  exp
JAN  exp            JSN  exp
JAP  exp            JSP  exp
JAM  exp            JSM  exp

R    exp

EX                  ERR
EX   exp            ERR  exp

PASS
```

BIT COUNT INSTRUCTIONS

```
Ai   PSj       Vi   PVj
Ai   QSj       Vi   QVj
Ai   ZSj
```

MONITOR OPERATIONS

```
CA,Aj  Ak        CCI
CL,Aj  Ak        ECI
CI,Aj            DCI

XA   Aj
RT   Sj
PCI  Sj
```

INTEGER ARITHMETIC OPERATIONS

```
Ai   Aj+Ak
Ai   Aj+1
Ai   Aj-Ak
Ai   Aj-1
Ai   Aj*Ak

Si   Sj+Sk     Vi   Sj+Vk     Vi   Vj+Vk
Si   Sj-Sk     Vi   Sj-Vk     Vi   Vj-Vk
```

INTER-REGISTER TRANSFERS

Ai	Ak	Si	Sk
Ai	-Ak	Si	-Sk
		Si	#Sk
Ai	Sj	Si	Ak
		Si	+Ak
		Si	+FAk
Ai	Bjk	Si	Tjk
Ai	CI	Si	Vj,Ak
Ai	CA,Aj	Si	VM
Ai	CE,Aj	Si	RT
Bjk	Ai	Tjk	Si
		Vi	Vk
		Vi	-Vk
		Vi,Ak	Sj
VL	Ak	VM	Sj
VL	1	VM	0

MEMORY TRANSFERS

(store)		(load)	
,A0	Bjk,Ai	Bjk,Ai	,A0
0,A0	Bjk,Ai	Bjk,Ai	0,A0
,A0	Tjk,Ai	Tjk,Ai	,A0
,A0	Tjk,Ai	Tjk,Ai	0,A0
exp,Ah	Ai	Ai	exp,Ah
exp,0	Ai	Ai	exp,0
exp,	Ai	Ai	exp,
,Ah	Ai	Ai	,Ah
exp,Ah	Si	Si	exp,Ah
exp,0	Si	Si	exp,0
exp,	Si	Si	exp,
,Ah	Si	Si	,Ah
,A0,Ak	Vj	Vi	,A0,Ak
,A0,1	Vj	Vi	,A0,1

REGISTER	VALUE
Ah, h=0	0
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}

LOGICAL OPERATORS	
&	0101
AND	1100
	0100
!	0101
OR	1100
	1101
\	0101
XOR	1100
	1001

A.3 CRAY X-MP SYMBOLIC MACHINE INSTRUCTIONS

LOGICAL OPERATIONS

Si	Sj&Sk	Vi	Sj&Vk	Vi	Vj&Vk
Si	Sj&SB				
Si	SB&Sj				
Si	#Sk&Sj				
Si	#SB&Sj				
Si	Sj!Sk	Vi	Sj!Vk	Vi	Vj!Vk
Si	Sj!SB				
Si	SB!Sj				
Si	Sj\Sk	Vi	Sj\Vk	Vi	Vj\Vk
Si	Sj\SB				
Si	SB\Sj				
Si	#Sj\Sk				
Si	#Sj\SB				
Si	#SB\Sj				
		VM	Vj,Z	Vi,VM	Vj,Z
		VM	Vj,N	Vi,VM	Vj,N
		VM	Vj,P	Vi,VM	Vj,P
		VM	Vj,M	Vi,VM	Vj,M
Si	Sj!Si&Sk				
Si	Sj!Si&SB	Vi	Sj!Vk&VM	Vi	Vj!Vk&VM
		Vi	#VM&Vk		

FLOATING-POINT OPERATIONS

EFI					
DFI					
Si	Sj+FSk	Vi	Sj+FVk	Vi	Vj+FVk
Si	+FSk	Vi	+FVk		
Si	Sj-FSk	Vi	Sj-FVk	Vi	Vj-FVk
Si	-FSk	Vi	-FVk		
Si	Sj*FSk	Vi	Sj*FVk	Vi	Vj*FVk
Si	Sj*HSk	Vi	Sj*HVk	Vi	Vj*HVk
Si	Sj*RSk	Vi	Sj*RVk	Vi	Vj*RVk
Si	Sj*ISk	Vi	Sj*IVk	Vi	Vj*IVk
Si	/HSj			Vi	/HVj

SHIFT INSTRUCTIONS

```
S0   Si<exp       S0   Si>exp
Si   Si<exp       Si   Si>exp

Si   Si,Sj<Ak     Si   Sj,Si>Ak
Si   Si,Sj<1      Si   Sj,Si>1
Si   Si<Ak        Si   Si>Ak

Vi   Vj<Ak        Vi   Vj>Ak
Vi   Vj<1         Vi   Vj>1

Vi   Vj,Vj<Ak     Vi   Vj,Vj>Ak
Vi   Vj,Vj<1      Vi   Vj,Vj>1
```

REGISTER ENTRY INSTRUCTIONS

```
Ah   exp          Si   <exp
Ai   -1           Si   #>exp
                  Si   >exp
Si   exp          Si   #<exp

Si   0            Si   SB
Si   1            Si   #SB
Si   -1
Si   1            Vi,Ak  0
Si   2            Vi   0
Si   4
Si   0.4          SMjk  1,TS
Si   0.6          SMjk  0
                  SMjk  1
```

PROGRAM BRANCHES AND EXITS

```
J    exp
J    Bjk

JAZ  exp          JSZ  exp
JAN  exp          JSN  exp
JAP  exp          JSP  exp
JAM  exp          JSM  exp

R    exp

EX                ERR
PASS
```

BIT COUNT INSTRUCTIONS

```
Ai   PSj          Vi   PVj
Ai   QSj          Vi   QVj
Ai   ZSj
```

MONITOR OPERATIONS

```
CA,Aj  Ak         CCI
CL,Aj  Ak         ECI
CI,Aj             DCI
MC,Aj             ERI
XA     Aj         DRI
RT     Sj         CLN  exp
PCI    Sj
SIPI   exp
SIPI
CIPI
```

INTEGER ARITHMETIC OPERATIONS

```
Ai   Aj+Ak
Ai   Aj+1
Ai   Aj-Ak
Ai   Aj-1
Ai   Aj*Ak

Si   Sj+Sk     Vi   Sj+Vk     Vi   Vj+Vk
Si   Sj-Sk     Vi   Sj-Vk     Vi   Vj-Vk
```

INTER-REGISTER TRANSFERS

A*i*	A*k*	S*i*	S*k*
A*i*	-A*k*	S*i*	-S*k*
		S*i*	#S*k*
A*i*	S*j*	S*i*	A*k*
		S*i*	+A*k*
A*i*	VL	S*i*	+FA*k*
A*i*	B*jk*	S*i*	T*jk*
A*i*	SB*j*	S*i*	ST*j*
A*i*	CI	S*i*	V*j*,A*k*
A*i*	CA,A*j*	S*i*	VM
A*i*	CE,A*j*	S*i*	RT
		S*i*	SM
		S*i*	SR*j*
B*jk*	A*i*	T*jk*	S*i*
SB*j*	A*i*	ST*j*	S*i*
		V*i*	V*k*
		V*i*	-V*k*
		V*i*,A*k*	S*j*
VL	A*k*	VM	S*j*
VL	1	VM	0
		SM	S*i*

MEMORY TRANSFERS

DBM
EBM
CMR

(store)		(load)	
,A0	B*jk*,A*i*	B*jk*,A*i*	,A0
0,A0	B*jk*,A*i*	B*jk*,A*i*	0,A0
,A0	T*jk*,A*i*	T*jk*,A*i*	,A0
0,A0	T*jk*,A*i*	T*jk*,A*i*	0,A0
exp,A*h*	A*i*	A*i*	*exp*,A*h*
exp,0	A*i*	A*i*	*exp*,0
exp,	A*i*	A*i*	*exp*,
,A*h*	A*i*	A*i*	,A*h*
exp,A*h*	S*i*	S*i*	*exp*,A*h*
exp,0	S*i*	S*i*	*exp*,0
exp,	S*i*	S*i*	*exp*,
,A*h*	S*i*	S*i*	,A*h*
,A0,A*k*	V*j*	V*i*	,A0,A*k*
,A0,1	V*j*	V*i*	,A0,1
,A0,V*k*	V*j*	V*i*	,A0,V*k*

REGISTER	VALUE
A*h*, *h*=0	0
A*i*, *i*=0	(A0)
A*j*, *j*=0	0
A*k*, *k*=0	1
S*i*, *i*=0	(S0)
S*j*, *j*=0	0
S*k*, *k*=0	2^{63}

LOGICAL OPERATORS	
&	0101
AND	$\underline{1100}$
	0100
!	0101
OR	$\underline{1100}$
	1101
\	0101
XOR	$\underline{1100}$
	1001

FUNCTIONAL INSTRUCTION SUMMARY B

This appendix contains an instruction summary, listed by function, for CRAY X-MP and CRAY-1 Computer Systems. A detailed description may be found on the referenced pages.

B.1 REGISTER ENTRY INSTRUCTIONS

Instructions in this category provide for entering values such as constants, expression values, or masks directly into registers.

B.1.1 ENTRIES INTO A REGISTERS

Machine Instruction	CAL	Description	Page
01*hijkm*†	A*h exp*	Transmit *ijkm* to A*h*, where the high-order bit of *i* is 1	3-23
020*ijkm* or 021*ijkm* or 022*ijk*	A*i exp*	Enter *exp* into A*i*	3-24
031*i*00††	A*i* -1	Enter -1 into A*i*	3-30

B.1.2 ENTRIES INTO S REGISTERS

Machine Instruction	CAL	Description	Page
040*ijkm* or 041*ijkm*	S*i exp*	Enter *exp* into S*i*	3-37
042*i*00††	S*i* -1	Enter -1 into S*i*	3-38

† CRAY X-MP Computer Systems only
†† Special CAL syntax

Machine Instruction	CAL	Description	Page
042*ijk*	S*i* <*exp*	Form ones mask in S*i* from right	3-38
042*ijk*†	S*i* #>*exp*	Form zeros mask in S*i* from left	3-38
042*i*77†	S*i* 1	Enter 1 into S*i*	3-38
043*i*00†	S*i* 0	Clear S*i*	3-38
043*ijk*	S*i* >*exp*	Form ones mask in S*i* from left	3-38
043*ijk*†	S*i* #<*exp*	Form zeros mask in S*i* from right	3-38
047*i*00†	S*i* #SB	Enter ones complement of sign bit in S*i*	3-41
051*i*00†	S*i* SB	Enter sign bit into S*i*	3-41
071*i*30	S*i* 0.6	Enter 0.75*(2**48) into S*i* as normalized floating-point constant	3-57
071*i*40	S*i* 0.4	Enter 0.5 into S*i* as normalized floating-point constant	3-57
071*i*50	S*i* 1.	Enter 1 into S*i* as normalized floating-point constant	3-57
071*i*60	S*i* 2.	Enter 2 into S*i* as normalized floating-point constant	3-57
071*i*70	S*i* 4.	Enter 4 into S*i* as normalized floating-point constant	3-57

B.1.3 ENTRIES INTO V REGISTERS

Machine Instruction	CAL	Description	Page
077*i*0*k*†	V*i*,A*k* 0	Clear element (A*k*) of register V*i*	3-62
145*iii*†	V*i* 0	Clear V*i*	3-67

† Special CAL syntax

B.1.4 ENTRIES INTO SEMAPHORE REGISTER

Machine Instruction	CAL	Description	Page
0034*jk*†	SM*jk* 1,TS	Test and set semaphore *jk*, $0 \le jk \le 31$ (decimal)	3-15
0036*jk*†	SM*jk* 0	Clear semaphore *jk*, $0 \le jk \le 31$ (decimal)	3-15
0037*jk*†	SM*jk* 1	Set semaphore *jk*, $0 \le jk \le 31$ (decimal)	3-15

B.2 INTER-REGISTER TRANSFER INSTRUCTIONS

Instructions in this group provide for transferring the contents of one register to another register. In some cases, the register contents can be complemented, converted to floating-point format, or sign extended as a function of the transfer.

B.2.1 TRANSFERS TO A REGISTERS

Machine Instruction	CAL	Description	Page
023*ij*0	A*i* S*j*	Transmit (S*j*) to A*i*	3-26
023*i*01†	A*i* VL	Transmit (VL) to A*i*	3-26
024*ijk*	A*i* B*jk*	Transmit (B*jk*) to A*i*	3-27
026*ij*7†	A*i* SB*j*	Transfer (SB*j*) to A*i*	3-28
030*i*0*k*††	A*i* A*k*	Transmit (A*k*) to A*i*	3-30
031*i*0*k*††	A*i* -A*k*	Transmit negative of (A*k*) to A*i*	3-30
033*i*00	A*i* CI	Channel number to A*i*	3-33
033*ij*0	A*i* CA,A*j*	Address of channel (A*j*) to A*i*	3-33
033*ij*1	A*i* CE,A*j*	Error flag of channel (A*j*) to A*i*	3-33

† CRAY X-MP Computer Systems only
†† Special CAL syntax

B.2.2 TRANSFERS TO S REGISTERS

Machine Instruction	CAL	Description	Page
025*ijk*	B*jk* A*i*	Transmit (A*i*) to B*jk*	3-27
027*ij*7†	SB*j* A*i*	Transfer (A*i*) to SB*j*	3-29
047*i*0*k*††	S*i* #S*k*	Transmit ones complement of (S*k*) to S*i*	3-41
051*i*0*k*††	S*i* S*k*	Transmit (S*k*) to S*i*	3-41
061*i*0*k*††	S*i* -S*k*	Transmit negative of (S*k*) to S*i*	3-50
071*i*0*k*	S*i* A*k*	Transmit (A*k*) to S*i* without sign extension	3-57
071*i*1*k*	S*i* +A*k*	Transmit (A*k*) to S*i* with sign extension	3-57
071*i*2*k*	S*i* +FA*k*	Transmit (A*k*) to S*i* as an unnormalized floating-point value	3-57
072*i*00	S*i* RT	Transmit (RTC) to S*i*	3-59
072*i*02†	S*i* SM	Read semaphore to S*i*	3-59
072*ij*3†	S*i* ST*j*	Read (ST*j*) register to S*i*	3-59
073*i*00	S*i* VM	Transmit (VM) to S*i*	3-59
073*ij*1†	S*i* SR*j*	Transfer (SR*j*) to S*j*; *j*=0	3-59
073*ij*3†	ST*j* S*i*	Transmit (S*i*) to ST*j*	3-59
074*ijk*	S*i* T*jk*	Transmit (T*jk*) to S*i*	3-59
075*ijk*	T*jk* S*i*	Transmit (S*i*) to T*jk*	3-59
076*ijk*	S*i* V*j*,A*k*	Transmit (V*j*, element (A*k*)) to S*i*	3-62

† CRAY X-MP Computer Systems only
†† Special CAL syntax

B.2.3 TRANSFERS TO V REGISTERS

Machine Instruction	CAL	Description	Page
077*ijk*	V*i*,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*)	3-62
142*i*0*k*†	V*i* V*k*	Transmit (V*k*) to V*i*	3-67
156*i*0*k*†	V*i* -V*k*	Transmit twos complement of (V*k*) to V*i*	3-77

B.2.4 TRANSFER TO VECTOR MASK REGISTER

Machine Instruction	CAL	Description	Page
0030*j*0	VM S*j*	Transmit (S*j*) to VM	3-15
003000†	VM 0	Clear VM	3-15

B.2.5 TRANSFER TO VECTOR LENGTH REGISTER

Machine Instruction	CAL	Description	Page
00200*k*	VL A*k*	Transmit (A*k*) to VL	3-11
002000†	VL 1	Enter 1 into VL	3-11

B.2.6 TRANSFER TO SEMAPHORE REGISTER

Machine Instruction	CAL	Description	Page
073*i*02††	SM S*i*	Load semaphores from S*i*	3-59

† Special CAL syntax
†† CRAY X-MP Computer Systems only

B.3 MEMORY TRANSFERS

This category contains instructions that transfer data between registers and memory, enable and disable concurrent block memory transfers, and assure completion of memory references.

B.3.1 BIDIRECTIONAL MEMORY TRANSFERS

Machine Instruction	CAL	Description	Page
002500†	DBM	Disable bidirectional memory transfers	3-13
002600†	EBM	Enable bidirectional memory transfers	3-13

B.3.2 MEMORY REFERENCES

Machine Instruction	CAL	Description	Page
002700†	CMR	Complete memory references	3-13

B.3.3 STORES

Machine Instruction	CAL	Description	Page
035*ijk*	,A0 B*jk*,A*i*	Store (A*i*) words starting at B*jk* to memory starting at (A0)	3-34
035*ijk*††	0,A0 B*jk*,A*i*	Store (A*i*) words starting at B*jk* to memory starting at (A0)	3-34
037*ijk*	,A0 T*jk*,A*i*	Store (A*i*) words starting at T*jk* to memory starting at (A0)	3-34
037*ijk*††	0,A0 T*jk*,A*i*	Store (A*i*) words starting at T*jk* to memory starting at (A0)	3-34

† CRAY X-MP Computer Systems only
†† Special CAL syntax

Machine Instruction	CAL	Description	Page
11*hijkm*	*exp*,A*h* A*i*	Store (A*i*) to (A*h*) + *exp*	3-63
11*hi*000†	,A*h* A*i*	Store (A*i*) to (A*h*)	3-63
110*ijkm*†	*exp*,0 A*i*	Store (A*i*) to *exp*	3-63
110*ijkm*†	*exp*, A*i*	Store (A*i*) to *exp*	3-63
13*hijkm*	*exp*,A*h* S*i*	Store (S*i*) to (A*h*) + *exp*	3-63
130*ijkm*†	*exp*,0 S*i*	Store (S*i*) to *exp*	3-63
130*ijkm*†	*exp*, S*i*	Store (S*i*) to *exp*	3-63
13*hi*000†	,A*h* S*i*	Store (S*i*) to (A*h*)	3-63
1770*jk*	,A0,A*k* V*j*	Store (V*j*) to memory starting at (A0) incremented by (A*k*)	3-89
1770*j*0†	,A0,1 V*j*	Store (V*j*) to a memory in consecutive addresses starting with (A0)	3-89
1771*jk*††	,A0,V*k* V*j*	Store (V*j*) to a memory using memory address (A0)+(V*k*)	3-89

B.3.4 LOADS

Machine Instruction	CAL	Description	Page
034*ijk*	B*jk*,A*i* ,A0	Read (A*i*) words starting at B*jk* from memory starting at (A0)	3-34
034*ijk*†	B*jk*,A*i* 0,A0	Read (A*i*) words starting at B*jk* from memory starting at (A0)	3-34
036*ijk*	T*jk*,A*i* ,A0	Read (A*i*) words starting at T*jk* from memory starting at (A0)	3-34
036*ijk*†	T*jk*,A*i* 0,A0	Read (A*i*) words starting at T*jk* from memory starting at (A0)	3-34

† Special CAL syntax
†† CRAY X-MP Computer Systems only

Machine Instruction	CAL	Description	Page
10*hijkm*	A*i* *exp*,A*h*	Read from ((A*h*) + *exp*) to A*i*	3-63
10*hi*000†	A*i* ,A*h*	Read from (A*h*) to A*i*	3-63
100*ijkm*†	A*i* *exp*,0	Read from (*exp*) to A*i*	3-63
100*ijkm*†	A*i* *exp*,	Read from (*exp*) to A*i*	3-63
12*hijkm*	S*i* *exp*,A*h*	Read from ((A*i*) + *exp*) to S*i*	3-63
120*ijkm*†	S*i* *exp*,0	Read from (*exp*) to S*i*	3-63
120*ijkm*†	S*i* *exp*	Read from (*exp*) to S*i*	3-63
12*hi*000†	S*i* ,A*h*	Read from (A*h*) to S*i*	3-63
176*i*0*k*	V*i* ,A0,A*k*	Read from memory starting at (A0) incremented by (A*k*) and load into V*i*	3-89
176*i*00†	V*i* ,A0,1	Read from consecutive memory addresses starting with (A0) and load into V*i*	3-89
176*i*1*k*††	V*i* ,A0,V*k*	Read from memory using memory address (A0) + (V*k*) and load into V*i*	3-89

B.4 INTEGER ARITHMETIC OPERATIONS

Integer arithmetic operations obtain operands from registers and return results to registers. No direct memory references are allowed.

The assembler recognizes several special syntax forms for increasing or decreasing register contents, such as the operands A*i*+1 and A*i*-1; however, these references actually result in register references such that the 1 becomes a reference to A*k* with *k*=0.

All integer arithmetic, whether 24-bit or 64-bit, is twos complement and is so represented in the registers. The Address Add functional unit and

† Special CAL syntax
†† CRAY X-MP Computer Systems only

Address Multiply functional unit perform 24-bit arithmetic. The Scalar Add functional unit and the Vector Add functional unit perform 64-bit arithmetic.

No overflow is detected by Integer functional units.

Multiplication of two fractional operands can be accomplished using the floating-point multiply instruction. The Floating-point Multiply functional unit recognizes the conditions where both operands have zero exponents as a special case and returns the high-order 48 bits of the result as an unnormalized fraction. Division of integers would require that they first be converted to floating-point format and then divided using the floating-point units.

B.4.1 24-BIT INTEGER ARITHMETIC

Machine Instruction	CAL	Description	Page
030*ijk*	A*i* A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*	3-30
030*ij*0†	A*i* A*j*+1	Integer sum of (A*j*) and 1 to A*i*	3-30
031*ijk*	A*i* A*j*-A*k*	Integer difference of (A*j*) less (A*k*) to A*i*	3-30
031*ij*0†	A*i* A*j*-1	Integer difference of (A*j*) less 1 to A*i*	3-30
032*ijk*	A*i* A*j**A*k*	Integer product of (A*j*) and (A*k*) to A*i*	3-32

B.4.2 64-BIT INTEGER ARITHMETIC

Machine Instruction	CAL	Description	Page
060*ijk*	S*i* S*j*+S*k*	Integer sum of (S*j*) and (S*k*) to S*i*	3-50
061*ijk*	S*i* S*j*-S*k*	Integer difference of (S*j*) less (S*k*) to S*i*	3-50

† Special CAL syntax

Machine Instruction	CAL	Description	Page
154*ijk*	V*i* S*j*+V*k*	Integer sums of (S*j*) and (V*k*) to V*i*	3-77
155*ijk*	V*i* V*j*+V*k*	Integer sums of (V*j*) and (V*k*) to V*i*	3-77
156*ijk*	V*i* S*j*-V*k*	Integer differences of (S*j*) and (V*k*) to V*i*	3-77
157*ijk*	V*i* V*j*-V*k*	Integer differences of (V*j*) less (V*k*) to V*i*	3-77

B.5 FLOATING-POINT ARITHMETIC

All floating-point arithmetic operations use registers as the source of operands and return results to registers.

Floating-point numbers are represented in a standard format throughout the CPU. This format is a packed representation of a binary coefficient and an exponent or power of 2. The coefficient is a 48-bit signed fraction. The sign of the coefficient is separated from the rest of the coefficient. Since the coefficient is signed magnitude, it is not complemented for negative values.

B.5.1 FLOATING-POINT RANGE ERRORS

Machine Instruction	CAL	Description	Page
002100	EFI	Enable floating-point interrupt	3-13
002200	DFI	Disable floating-point interrupt	3-13

B.5.2 FLOATING-POINT ADDITION AND SUBTRACTION

Machine Instruction	CAL	Description	Page
062*ijk*	S*i* S*j*+FS*k*	Floating-point sum of (S*j*) and (S*k*) to S*i*	3-51

Machine Instruction	CAL	Description	Page
062*i*0*k*†	S*i* +FS*k*	Normalize (S*k*) to S*i*	3-51
063*ijk*	S*i* S*j*-FS*k*	Floating-point difference of (S*j*) less (S*k*) to S*i*	3-51
063*i*0*k*†	S*i* -FS*k*	Transmit the negative of (S*k*) as a normalized floating-point value	3-51
170*ijk*	V*i* S*j*+FV*k*	Floating-point sums of (S*j*) and (V*k*) to V*i*	3-81
170*i*0*k*†	V*i* +FV*k*	Normalize (V*k*) to V*i*	3-81
171*ijk*	V*i* V*j*+FV*k*	Floating-point sums of (V*j*) (V*k*) to V*i*	3-81
172*ijk*	V*i* S*j*-FV*k*	Floating-point differences of (S*j*) less (V*k*) to V*i*	3-81
172*i*0*k*†	V*i* -FV*k*	Transmit normalized negative of (V*k*) to V*i*	3-81
173*ijk*	V*i* V*j*-FV*k*	Floating-point differences of (V*j*) less (V*k*) to V*i*	3-81

B.5.3 FLOATING-POINT MULTIPLICATION

Machine Instruction	CAL	Description	Page
064*ijk*	S*i* S*j**FS*k*	Floating-point product of (S*j*) and (S*k*) to S*i*	3-53
065*ijk*	S*i* S*j**HS*k*	Half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*	3-53
066*ijk*	S*i* S*j**RS*k*	Rounded floating-point product of (S*j*) and (S*k*) to S*i*	3-53

† Special CAL syntax

Machine Instruction	CAL	Description	Page
160*ijk*	V*i* S*j**FV*k*	Floating-point products of (S*j*) and (V*k*) to V*i*	3-79
161*ijk*	V*i* V*j**FV*k*	Floating-point products of (V*j*) and (V*k*) to V*i*	3-79
162*ijk*	V*i* S*j**HV*k*	Half-precision rounded floating-point products of (S*j*) and (V*k*) to V*i*	3-79
163*ijk*	V*i* V*j**HV*k*	Half-precision rounded floating-point products of (V*j*) and (V*k*) to V*i*	3-79
164*ijk*	V*i* S*j**RV*k*	Rounded floating-point products of (S*j*) and (V*k*) to V*i*	3-79
165*ijk*	V*i* V*j**RV*k*	Rounded floating-point products of (V*j*) and (V*k*) to V*i*	3-79

B.5.4 RECIPROCAL ITERATION

Machine Instruction	CAL	Description	Page
067*ijk*	S*i* S*j**IS*k*	2-floating-point product of (S*j*) and (S*k*) to S*i*	3-53
166*ijk*	V*i* S*j**IV*k*	2-floating-point products of (S*j*) and (V*k*) to V*i*	3-79
167*ijk*	V*i* V*j**IV*k*	2-floating-point products of (V*j*) and (V*k*) to V*i*	3-79

B.5.5 RECIPROCAL APPROXIMATION

Machine Instruction	CAL	Description	Page
070*ij*0	S*i* /HS*j*	Floating-point reciprocal approximation of (S*j*) to S*i*	3-55
174*ij*0	V*i* /HV*j*	Floating-point reciprocal approximation of (V*j*) to V*i*	3-84

B.6 LOGICAL OPERATIONS

The Scalar and Vector Logical functional units perform bit-by-bit manipulation of 64-bit quantities. Operations provide for logical products, logical differences, logical sums, logical equivalence, and merges.

A logical product (& operator) is the AND function.

A logical difference (\ operator) is the EXCLUSIVE OR function.

A logical sum (! operator) is the INCLUSIVE OR function.

A logical merge combines two operands depending on a ones mask in a third operand. The result is defined by (operand 2 & mask)!(operand 1 & #mask).

B.6.1 LOGICAL PRODUCTS

Machine Instruction	CAL	Description	Page
044*ijk*	S*i* S*j*&S*k*	Logical products of (S*j*) and (S*k*) to S*i*	3-40
044*ij*0†	S*i* S*j*&SB	Sign bit of (S*j*) to S*i*	3-40
044*ij*0†	S*i* SB&S*j*	Sign bit of (S*j*) to S*i*; *j*≠0	3-40
045*ijk*	S*i* #S*k*&S*j*	Logical product of (S*j*) and #(S*k*) to S*i*	3-40
045*ij*0†	S*i* #SB&S*j*	(S*j*) with sign bit cleared to S*i*	3-40
140*ijk*	V*i* S*j*&V*k*	Logical products of (S*j*) and (V*k*) to V*i*	3-67
141*ijk*	V*i* V*j*&V*k*	Logical products of (V*j*) and (V*k*) to V*i*	3-67

† Special CAL syntax

B.6.2 LOGICAL SUMS

Machine Instruction	CAL	Description	Page
051*ijk*	S*i* S*j*!S*k*	Logical sum of (S*j*) and (S*k*) to S*i*	3-41
051*ij*0†	S*i* S*j*!SB	Logical sum of (S*j*) and sign bit to S*i*	3-41
051*ij*0†	S*i* SB!S*j*	Logical sum of sign bit and (S*j*) to S*i*; *j*≠0	3-41
142*ijk*	V*i* S*j*!V*k*	Logical sums of (S*j*) and (V*k*) to V*i*	3-67
143*ijk*	V*i* V*j*!V*k*	Logical sums of (V*j*) and (V*k*) to V*i*	3-67

B.6.3 LOGICAL DIFFERENCES

Machine Instruction	CAL	Description	Page
046*ijk*	S*i* S*j*\S*k*	Logical differences of (S*j*) and (S*k*) to S*i*	3-40
046*ij*0†	S*i* S*j*\SB	Enter (S*j*) into S*i* with sign bit toggled	3-40
046*ij*0†	S*i* SB\S*j*	Enter (S*j*) into S*i* with sign bit toggled; *j*≠0	3-40
144*ijk*	V*i* S*j*\V*k*	Logical differences of (S*j*) and (V*k*) to V*i*	3-67
145*ijk*	V*i* V*j*\V*k*	Logical differences of (V*j*) and (V*k*) to V*i*	3-67

† Special CAL syntax

B.6.4 LOGICAL EQUIVALENCE

Machine Instruction	CAL	Description	Page
047*ijk*	S*i* #S*j*\S*k*	Logical equivalence of (S*j*) and (S*k*) to S*i*	3-40
047*ij*0†	S*i* #S*j*\SB	Logical equivalence of (S*j*) and sign bit to S*i*	3-40
047*ij*0†	S*i* #SB\S*j*	Logical equivalence of sign bit and (S*j*) to S*i*; *j*≠0	3-40

B.6.5 VECTOR MASK

Machine Instruction	CAL	Description	Page
1750*j*0	VM V*j*,Z	Set VM bits for zero elements of V*j*	3-87
1750*j*1	VM V*j*,N	Set VM bits for nonzero elements of V*j*	3-87
1750*j*2	VM V*j*,P	Set VM bits for positive elements of V*j*	3-87
1750*j*3	VM V*j*,M	Set VM bits for negative elements of V*j*	3-87
175*ij*4††	V*i*,VM V*j*,Z	Set VM bits and register V*i* to V*j*, for zero elements of V*j*	3-87
175*ij*5††	V*i*,VM V*j*,N	Set VM bits and register V*i* to V*j*, for nonzero elements of V*j*	3-87
175*ij*6††	V*i*,VM V*j*,P	Set VM bits and register V*i* to V*j*, for positive elements of V*j*	3-87
175*ij*7††	V*i*,VM V*j*,M	Set VM bits and register V*i* to V*j*, for negative elements of V*j*	3-87

† Special CAL syntax
†† CRAY X-MP Computer Systems only

B.6.6 MERGE

Machine Instruction	CAL	Description	Page
050*ijk*	S*i* S*j*!S*i*&S*k*	Scalar merge of (S*i*) and (S*j*) to S*i*	3-41
050*ij*0†	S*j*!S*i*&SB	Scalar merge of (S*i*) and sign bit of (S*j*) to S*i*	3-41
146*ijk*	V*i* S*j*!V*k*&VM	Vector merge of (S*j*) and (V*k*) to V*i*	3-67
146*i*0*k*†	V*i* #VM&V*k*	Vector merge of (V*k*) and zero to V*i*	3-67
147*ijk*	V*i* V*j*!V*k*&VM	Vector merge of (V*j*) and (V*k*) to V*i*	3-67

B.7 SHIFT INSTRUCTIONS

The Scalar Shift functional unit and Vector Shift functional unit shift 64-bit quantities or 128-bit quantities. A 128-bit quantity is formed by concatenating two 64-bit quantities. The number of bits a value is shifted left or right is determined by the value of an expression for some instructions and by the contents of an A register for other instructions. If the count is specified by an expression, the value of the expression must not exceed 64.

Machine Instruction	CAL	Description	Page
052*ijk*	S0 S*i*<*exp*	Shift (S*i*) left *exp* places to S0	3-46
053*ijk*	S0 S*i*>*exp*	Shift (S*i*) right *exp* places to S0	3-46
054*ijk*	S*i* S*i*<*exp*	Shift (S*i*) left *exp* places to S*i*	3-46
055*ijk*	S*i* S*i*>*exp*	Shift (S*i*) right *exp* places to S*i*	3-46
056*ijk*	S*i* S*i*,S*j*<A*k*	Left shift by (A*k*) of (S*i*) and (S*j*) to S*i*	3-48

† Special CAL syntax

Machine Instruction	CAL	Description	Page
056*ij*0†	S*i* S*i*,S*j*<1	Left shift by one of (S*i*) and (S*j*) to S*i*	3-48
056*i*0*k*†	S*i* S*i*<A*k*	Left shift by (A*k*) of (S*i*) to S*i*	3-48
057*ijk*	S*i* S*j*,S*i*>A*k*	Right shift by (A*k*) of (S*j*) and (S*i*) to S*i*	3-48
057*ij*0†	S*i* S*j*,S*i*>1	Right shift by one of (S*j*) and (S*i*) to S*i*	3-48
057*i*0*k*†	S*i* S*i*>A*k*	Right shift by (A*k*) of (S*i*) to S*i*	3-48
150*ijk*	V*i* V*j*<A*k*	Shift (V*j*) left (A*k*) places to V*i*	3-72
150*ij*0†	V*i* V*j*<1	Shift (V*j*) left one place to V*i*	3-72
151*ijk*	V*i* V*j*>A*k*	Shift (V*j*) right (A*k*) places to V*i*	3-72
151*ij*0†	V*i* V*j*>1	Shift (V*j*) right one place to V*i*	3-72
152*ijk*	V*i* V*j*,V*j*<A*k*	Double shift (V*j*) left (A*k*) places to V*i*	3-74
152*ij*0†	V*i* V*j*,V*j*<1	Double shift (V*j*) left one place to V*i*	3-74
153*ijk*	V*i* V*j*,V*j*>A*k*	Double shift (V*j*) right (A*k*) places to V*i*	3-74
153*ij*0†	V*i* V*j*,V*j*>1	Double shift (V*j*) right one place to V*i*	3-74

B.8 BIT COUNT INSTRUCTIONS

The instructions described in this category provide for counting the number of bits in an S or V register or counting the number of leading 0 bits in an S or V register.

† Special CAL syntax

B.8.1 SCALAR POPULATION COUNT

Machine Instruction	CAL	Description	Page
026*ij*0	A*i* PS*j*	Population count of (S*j*) to A*i*	3-28

B.8.2 VECTOR POPULATION COUNT

Machine Instruction	CAL	Description	Page
174*ij*1†	V*i* PV*j*	Population count of (V*j*) to (V*i*)	3-86

B.8.3 SCALAR POPULATION COUNT PARITY

Machine Instruction	CAL	Description	Page
026*ij*1†	A*i* QS*j*	Population count parity of (S*j*) to A*i*	3-28
174*ij*2†	V*i* QV*j*	Population count parity of (V*j*) to (V*i*)	3-86

B.8.4 SCALAR LEADING ZERO COUNT

Machine Instruction	CAL	Description	Page
027*ij*0	A*i* ZS*j*	Leading zero count of (S*j*) to A*i*	3-29

B.9 BRANCH INSTRUCTIONS

Instructions in this category include conditional and unconditional branch instructions. An expression or the contents of a B register specify the branch address. An address is always taken to be a parcel address when the instruction is executed. If an expression has a word-address attribute, the assembler issues an error message.

† Optional on CRAY-1 (Models A and B)

B.9.1 UNCONDITIONAL BRANCH INSTRUCTIONS

Machine Instruction	CAL	Description	Page
0050*jk*	J B*jk*	Jump to (B*jk*)	3-18
006*ijkm*	J *exp*	Jump to *exp*	3-19

B.9.2 CONDITIONAL BRANCH INSTRUCTIONS

Machine Instruction	CAL	Description	Page
010*ijkm*	JAZ *exp*	Branch to *exp* if (A0)=0	3-21
011*ijkm*	JAN *exp*	Branch to *exp* if (A0)≠0	3-21
012*ijkm*	JAP *exp*	Branch to *exp* if (A0) positive	3-21
013*ijkm*	JAM *exp*	Branch to *exp* if (A0) negative	3-21
014*ijkm*	JSZ *exp*	Branch to *exp* if (S0)=0	3-22
015*ijkm*	JSN *exp*	Branch to *exp* if (S0)≠0	3-22
016*ijkm*	JSP *exp*	Branch to *exp* if (S0) positive	3-22
017*ijkm*	JSM *exp*	Branch to *exp* if (S0) negative	3-22

B.9.3 RETURN JUMP

Machine Instruction	CAL	Description	Page
001000†	PASS	Pass	3-3
007*ijkm*	R *exp*	Return jump to *exp*; set B00 to (P)+2	3-20

† Special CAL syntax

B.9.4 NORMAL EXIT

Machine Instruction	CAL	Description	Page
004000	EX	Normal exit	3-17
004*ijk*†	EX *exp*	Normal exit	3-17

B.9.5 ERROR EXIT

Machine Instruction	CAL	Description	Page
000000	ERR	Error exit	3-2
000*ijk*†	ERR *exp*	Error exit	3-2

B.10 MONITOR INSTRUCTIONS

Instructions described in this category are executed only when the CPU is in monitor mode. An attempt to execute one of these instructions when not in monitor mode is treated as a no-op.

The instructions perform specialized functions useful to the operating system.

B.10.1 CHANNEL CONTROL

Machine Instruction	CAL	Description	Page
0010*jk*	CA,A*j* A*k*	Set the Current Address (CA) register, indicated by (A*j*), to (A*k*) and activate the channel	3-3
0011*jk*	CL,A*j* A*k*	Set the channel (A*j*) limit address to (A*k*)	3-4

† Special CAL syntax on CRAY-1 Computer Systems only

Machine Instruction	CAL	Description	Page
0012*j*0	CI,A*j*	Clear Channel (A*j*) Interrupt flag	3-5
0012*j*1[†]	MC,A*j*	Clear Channel (A*j*) Interrupt flag and Error flag; set device master-clear (output channel); clear device ready-held (input channel)	3-5
0013*j*0	XA A*j*	Enter XA register with (A*j*)	3-6

B.10.2 SET REAL-TIME CLOCK

Machine Instruction	CAL	Description	Page
0014*j*0	RT S*j*	Enter RTC with (S*j*)	3-7

B.10.3 PROGRAMMABLE CLOCK INTERRUPT INSTRUCTIONS[††]

Machine Instruction	CAL	Description	Page
0014*j*4	PCI S*j*	Set program interrupt interval	3-7
001405	CCI	Clear clock interrupt	3-7
001406	ECI	Enable clock interrupts	3-7
001407	DCI	Disable clock interrupts	3-7

B.10.4 INTERPROCESSOR INTERRUPT INSTRUCTIONS[†]

Machine Instruction	CAL	Description	Page
0014*j*1	SIPI *exp*	Set interprocessor interrupt request of CPU *exp*; $0 \le exp \le 3$	3-7

† CRAY X-MP Computer Systems only
†† Optional on CRAY-1 (Models A and B)

Machine Instruction	CAL	Description	Page
001401†	SIPI	Set interprocessor interrupt request	3-7
001402	CIPI	Clear interprocessor interrupt	3-7

B.10.5 CLUSTER NUMBER INSTRUCTIONS††

Machine Instruction	CAL	Description	Page
0014*j*3	CLN *exp*	Cluster number = *exp*	3-7

B.10.6 OPERAND RANGE ERROR INTERRUPT INSTRUCTIONS††

Machine Instruction	CAL	Description	Page
002300	ERI	Enable interrupt on (address) range error	3-13
002400	DRI	Disable interrupt on (address) range error	3-13

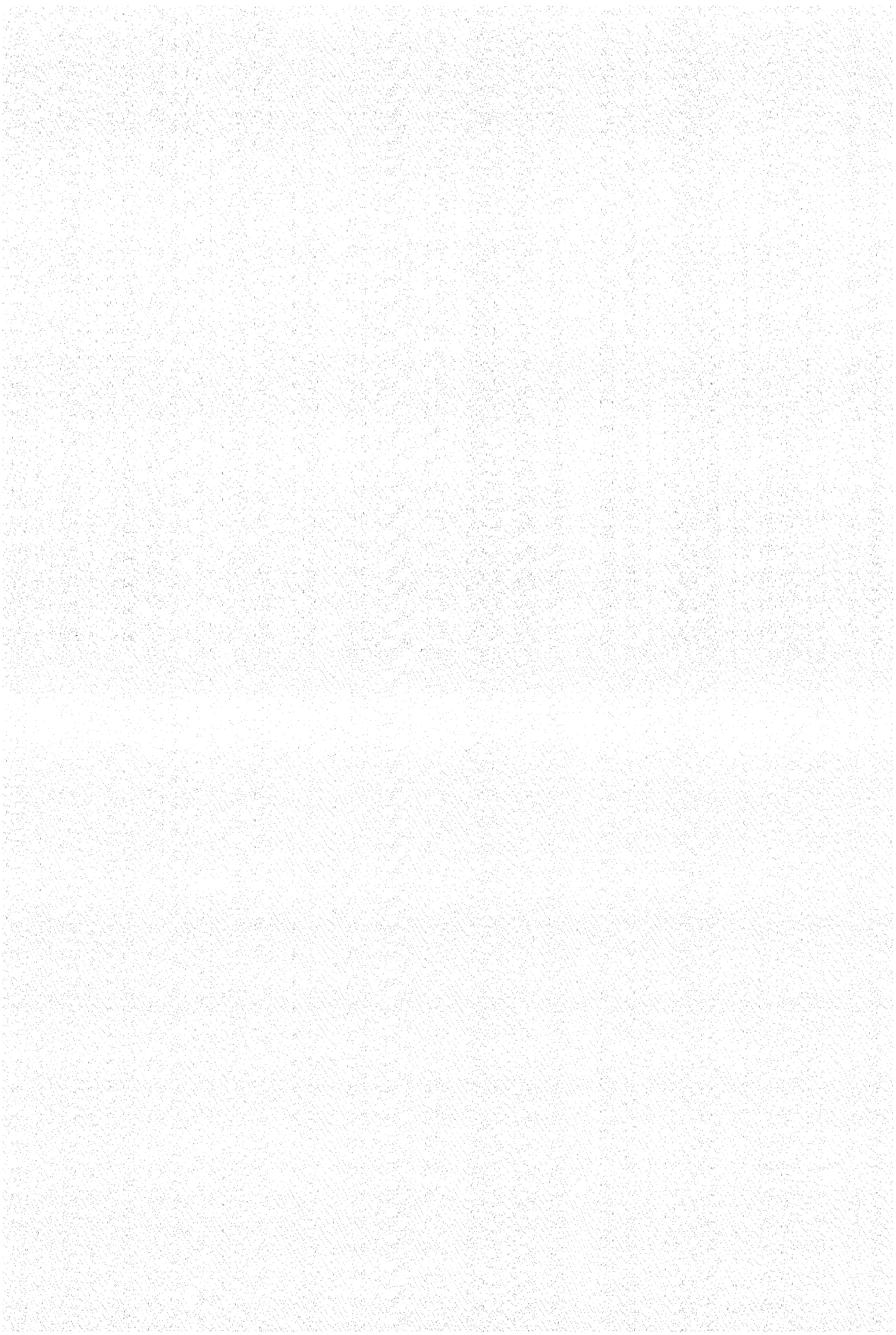
B.10.7 PERFORMANCE COUNTERS†† †††

Machine Instruction	CAL	Description	Page
0015*j*0		Select performance monitor	3-10
001501		Set maintenance read mode	3-10
001511		Load diagnostic checkbyte with S1	3-10
001521		Set maintenance write mode 1	3-10

† Special CAL syntax
†† CRAY X-MP Computer Systems only
††† Instructions not supported by CAL at this time

Machine Instruction	CAL	Description	Page
001531		Set maintenance write mode 2	3-10
073*i*11		Read performance counter into S*i*	3-59
073*i*21		Increment performance counter	3-59
073*i*31		Clear all maintenance modes	3-59

INDEX



INDEX

READER COMMENT FORM

CRAY X-MP and CRAY-1 Symbolic Machine Instructions Reference Manual SR-0085

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME ____________________

JOB TITLE ____________________

FIRM ____________________

ADDRESS ____________________

CITY __________ STATE ______ ZIP ______

DATE ____________________

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention:
PUBLICATIONS

2520 Pilot Knob Road
Suite 350
Mendota Heights, MN 55120
U.S.A.

STAPLE

READER COMMENT FORM

CRAY X-MP and CRAY-1 Symbolic Machine Instructions Reference Manual SR-0085

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME ______________________________

JOB TITLE ______________________________

FIRM ______________________________

ADDRESS ______________________________

CITY ____________ STATE ______ ZIP ______

DATE ______________________________

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention:
PUBLICATIONS

2520 Pilot Knob Road
Suite 350
Mendota Heights, MN 55120
U.S.A.