Instruction Set Overview
-115-A

Instruction Set Overview

(CRAY T90™ Series)

HTM-115-A

Cray Research Proprietary

Cray Research, Inc.

Record of Revision

REVISION	DESCRIPTION
	March 1995. Original printing.
A	September 1995. Revision A incorporates minor technical corrections to the instruction set.

Any shipment to a country outside of the United States requires a letter of assurance from Cray Research, Inc.

Requests for copies of Cray Research, Inc. publications should be directed to:

CRAY RESEARCH, INC.
Logistics
6251 South Prairie View Road
Chippewa Falls, WI 54729

Comments about this publication should be directed to:

CRAY RESEARCH, INC.
Service Publications and Training
890 Industrial Blvd.
Chippewa Falls, WI 54729

INSTRUCTION SET OVERVIEW

Figures

Tables

This overview describes the CPU instruction set. Depending on the state of the Triton mode (TRI) bit in the exchange package, the CPU operates in one of two modes: Triton mode or C90 mode.

In Triton mode, the A registers are 64 bits wide; bit 63 is the sign bit. (Software written for earlier systems needs to be recompiled before it can run in Triton mode.) In C90 mode, the CRAY T90 series system is binary-compatible with software written for the CRAY C90 series computer system. The A registers are 32 bits wide; bit 31 is the sign bit.

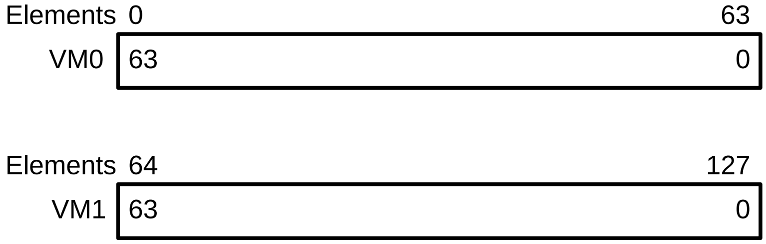
Some instructions operate differently in Triton mode than in C90 mode; the following subsections explain these differences.

Notational Conventions

This document uses the following conventions:

- Machine instructions are octal; all other numbers are decimal unless otherwise indicated.
- Register bits are numbered from right to left.
- The letter n represents a specified value.
- Variable parameters are in *italic* type.
- The symbol * designates an arithmetic product.
- The VM register contains the vector mask bits, which consist of two parts: VM0 and VM1. As shown in Figure 1, VM0 contains vector mask bits for elements 0 through 63; VM1 contains vector mask bits for elements 64 through 127.

Figure 1. Vector Element Layout



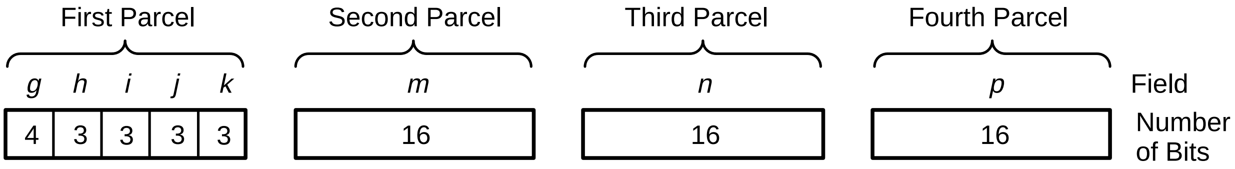
Instruction Formats

Instructions can be 1 parcel (16 bits), 3 parcels (48 bits), or 4 parcels (64 bits) long. Instructions contain 4 parcels per word. Within a word, parcels are numbered 0 through 3 from left to right.

A 3- or 4-parcel instruction can begin in any parcel of a word and can span a word boundary. For example, a 3-parcel instruction beginning in parcel 3 of a word ends in parcel 1 of the next word. No padding of word boundaries is required. Any parcel position can be addressed in branch instructions.

Figure 2 shows the general instruction format. The first parcel is divided into five fields. The second, third, and fourth parcels each contain a single field. Figure 4 and Figure 5 show how multiparcel instructions are actually stored in memory.

Figure 2. General Instruction Format



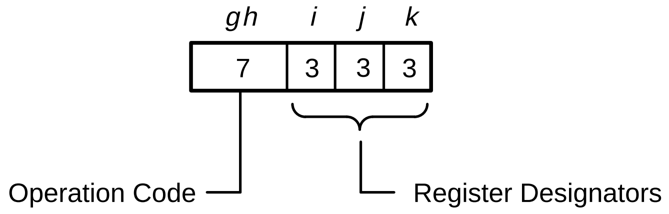
One-parcel Instruction Formats

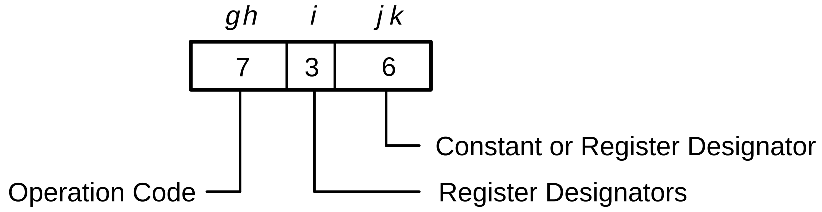
Most instructions are 1-parcel instructions; there are two types of 1-parcel instruction formats as shown in the following list. Figure 3 illustrates these two formats.

- 1-parcel instructions with discrete *j* and *k* fields
- 1-parcel instructions with combined *j* and *k* fields

In 1-parcel instructions with discrete *j* and *k* fields, the *j* and *k* fields usually designate operand registers. The *i* field designates a destination register. Some instructions do not use all three of these fields. Other instructions use the *i* or *k* field to provide additional bits for the operation code.

Figure 3. One-parcel Instruction Formats





In 1-parcel instructions with combined *j* and *k* fields, the *jk* field usually contains a constant or designates a source or destination register. The *i* field usually designates a destination or source register. Some instructions use the *i* field or bit 2 of the *j* field to provide additional bits for the operation code.

Some 1-parcel instructions of both formats are part of the extended instruction set. For example, they perform different operations when immediately preceded by the extended instruction set (EIS) instruction 005400.

Three-parcel Instruction Formats

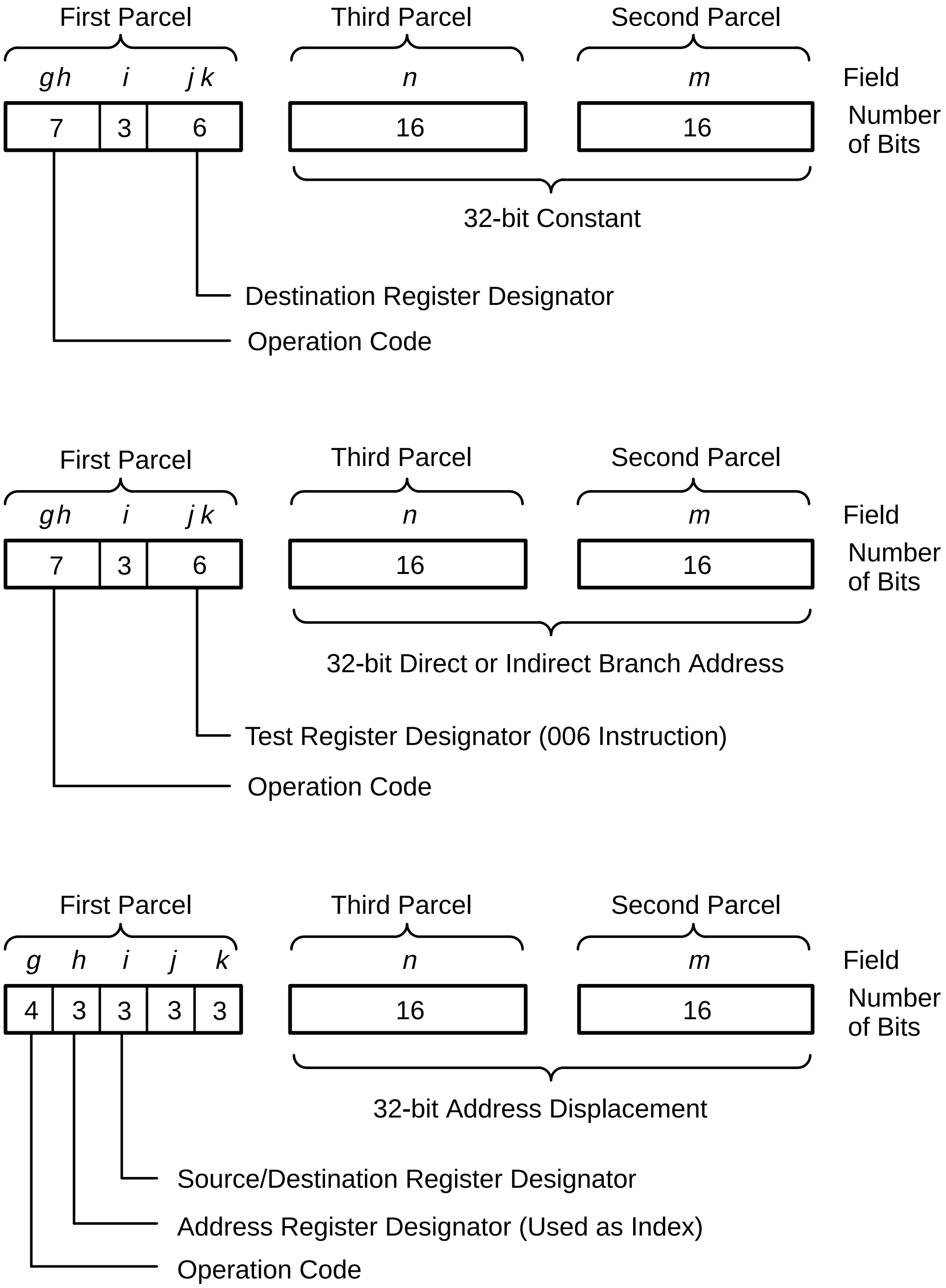
Some instructions are 3-parcel instructions. Figure 4 shows the 3-parcel format.

- 3-parcel instruction with field *nm* as a constant
- 3-parcel instruction with field *nm* as a branch address
- 3-parcel instruction with field *nm* as an address displacement

In all three formats, field *nm* is a 32-bit field with parcel *n* (the last parcel of the instruction) the most significant parcel.

Three-parcel instructions with the *nm* fields as constants transmit a constant value to an A or S register (instructions 020, 021, 040, and 041). The *i* field specifies the destination register. The *j* and *k* fields are not used, except that bits 1 and 2 of the *j* field specify different operations for instructions 020 and 040.

Figure 4. Three-parcel Instruction Formats



Three-parcel instructions with the *nm* fields as jump addresses are used for all types of jumps (instructions 006 through 017). Instructions 006 and 007 use *i* field bit 0 to distinguish between direct and indirect jumps.

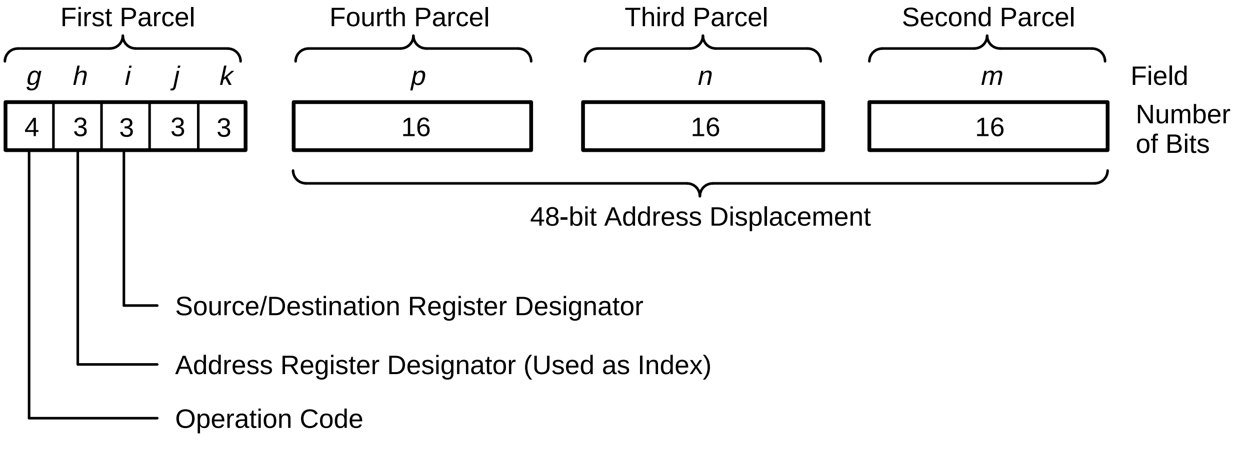
Instruction 006 uses *i* field bit 2 to distinguish between unconditional and conditional jumps. For conditional jumps, instruction 006 uses the *jk* field as the test register designator. Instructions 010 through 017 do not use the *i*, *j*, and *k* fields.

Three-parcel instructions with field *nm* as address displacements are used for A-register and S-register memory references (instructions 10*h* through 13*h*) using normal addressing. The *h* field selects an A register to be used as an address index. The *i* field designates an A or S register as the source or destination of the data. For memory read references (instructions 10*h* and 12*h*) *j* field bit 1 disables/enables bypass of the data cache. Bit 2 of the *j* field must be 0 to indicate a 3-parcel (normal addressing) instruction. The *k* field is not used.

Four-parcel Instruction Format

Figure 5 shows the 4-parcel instruction format. Field *pnm* is a 48-bit field with parcel *p* (the last parcel of the instruction) as the most significant parcel.

Figure 5. Four-Parcel Instruction Formats



Four-parcel instructions are used for A- and S-register memory references (instructions 10*h* through 13*h*) that use extended addressing. The *h* field selects an A register to be used as an address index. The *i* field designates an A or S register as the source or destination of the data. For memory read references (instructions 10*h* and 12*h*), *j* field bit 1 disables/enables bypass of the data cache. Bit 2 of the *j* field must be 1 to indicate a 4-parcel (extended addressing) instruction. The *k* field is not used.

Extended Instruction Set

The operation of some 1-parcel instructions is modified when they immediately follow a special instruction parcel (005400). The set of modified instructions is called the extended instruction set (EIS).

Each EIS instruction must be immediately preceded by the instruction parcel 005400 or the instruction performs its normal operation. For example, if instruction 044*ijk* is *not* preceded by parcel 005400, it computes the logical sum of registers S*j* and S*k* and transmits the result to register S*i*. If instruction 044*ijk is* preceded by parcel 005400, it computes the logical sum of registers A*j* and A*k* and transmits the result to register A*i*.

Special Register Values

If register A0 or S0 is referenced in the *h*, *j*, or *k* field of certain instructions, the contents of the respective register are 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 special operand does not alter the actual value of the A0 or S0 register. If register A0 or S0 is used in the *i* field as the operand, the actual value of the register is provided. Cray Assembly Language (CAL) issues a caution-level error message for A0 or S0 when 0 does not apply to the *i* field. Table 1 lists the special register values.

Table 1. Special Register Values

Instruction Field	Operand Value
A*h*, *h* = 0	0
A*j*, *j* = 0	0
A*k*, *k* = 0	1
S*j*, *j* = 0	0
S*k*, *k* = 0	bit 63 = 1

Undefined Instructions

Executing an illegal instruction produces undefined results. Some instructions cause an error exit, others are no-operation (no-op) instructions, etc. However, no illegal instruction will halt or hang the CPU.

Triton-mode Instructions

Triton mode is active when the Triton-mode (TRI) bit in the exchange package modes field is set. Some instructions execute correctly only if the CPU is operating in Triton mode. If a Triton-mode instruction issues while the CPU is operating in C90 mode, the result is undefined. Table 2 lists the instructions that are privileged to Triton mode.

Table 2. Triton-mode Instructions

Machine Instruction	CAL Syntax		Instruction Type
0030*j*2	VM0	A*j*	Instructions that require 64-bit A registers.
0030*j*3	VM1	A*j*	
020*i*20*nm*	A*i*	A*i*:*exp*	
020*i*40*nm*	A*i*	*exp*:A*i*	
027*ij*1	A*i*	ZA*j*	
005400 042*ijk*	A*i*	<*exp*	
005400 043*ijk*	A*i*	>*exp*	
005400 044*ijk*	A*i*	A*j*&A*k*	
005400 045*ijk*	A*i*	#A*k*&A*j*	
005400 046*ijk*	A*i*	A*j*\A*k*	
005400 047*ijk*	A*i*	#A*j*\A*k*	
005400 050*ijk*	A*i*	A*j*!A*i*&A*k*	
005400 051*ijk*	A*i*	A*j*!A*k*	
005400 052*ijk*	A0	A*i*<*exp*	
005400 053*ijk*	A0	A*i*>*exp*	
005400 054*ijk*	A*i*	A*i*<*exp*	
005400 055*ijk*	A*i*	A*i*>*exp*	
005400 056*ijk*	A*i*	A*i*,A*j*<A*k*	
005400 057*ijk*	A*i*	A*j*,A*i*>A*k*	
073*i*20	A*i*	VM0	
073*i*30	A*i*	VM1	
10*hi*40*pnm*	A*i*	*exp*,A*h*	A- and S-register memory reference instructions that use extended addressing.
10*hi*60*pnm*	A*i*	*exp*,A*h*,BC	
11*hi*40*pnm*	*exp*,A*h*	A*i*	
12*hi*40*pnm*	S*i*	*exp*,A*h*	
12*hi*60*pnm*	S*i*	*exp*,A*h*,BC	
13*hi*40*pnm*	*exp*,A*h*	S*i*	

Machine Instruction	CAL Syntax		Instruction Type
006100*nm* 007100*nm*	IJ IR	*exp* *exp*	Indirect jump and indirect return jump instructions.
005400 153*ij*0 005400 153*ij*1 005400 176*ijk*	V*i* V*i*,[VM] V*i*:V*j*	V*j*,[VM] V*j* ,A0:A*k*,V*k*	Vector compress, expand, and double gather instructions.
001501	–		Clear performance monitor pointer.

Monitor-mode Instructions

Monitor mode is active when the monitor mode (MM) bit in the exchange package modes field is set.

Monitor-mode instructions perform specialized functions that are useful to the operating system. These instructions execute normally only if the CPU is in monitor mode. If a monitor-mode instruction issues while the CPU is in user mode, the instruction is treated as a no-op instruction. However, all hold-issue conditions still apply.

In normal user mode, most monitor-mode instructions act as simple no-ops; program execution continues with the next sequential instruction. Instruction 073*ij*1 (*j* = 2 through 7) is the only exception. If this instruction is executed in normal user mode, it returns a value of 0 to register S*i*.

In interrupt-on-monitor-instruction (IMI) mode, most monitor-mode instructions execute as no-ops, but a monitor instruction interrupt (MII) occurs before the next instruction issues. Instruction 073*ij*1 (*j* = 2 through 7) is the only monitor-mode instruction that executes normally when the IMI mode bit is set. Table 3 lists the instructions that are privileged to monitor mode.

Table 3. Monitor-mode Instructions

Machine Instruction	CAL Syntax	Machine Instruction	CAL Syntax
0010*jk* (*jk* ≠ 0)	CA,A*j* A*k*	001406	ECI
0011*jk*	CL,A*j* A*k*	001407	DCI
0012*j*0	CI,A*j*	001500	—
0012*j*1	MC,A*j*	001501	—
0012*j*2	DI,A*j*	001600	ESI
0012*j*3	EI,A*j*	001640	BCD
0013*j*0	XA A*j*	0017*jk*	BP*k* A*j*
0013*j*1	A*j* XA	023*ij*6	A*i* EA,*j*
001302	EMI	023*ij*7	A*i* EA,A*j*
001303	DMI	027*ij*2	EA*j* A*i*
0014*j*0	RT S*j*	027*ij*3	EA,A*j* A*i*
0014*j*1	SIPI A*j*	033*i*00	A*i* CI
001402	CIPI	033*ij*0 (*j* ≠ 0)	A*i* CA,A*j*
0014*j*3	CLN A*j*	033*ij*1 (*j* ≠ 0)	A*i* CE,A*j*
0014*j*4	PCI S*j*	073*ij*1 (*j* = 2 – 7)	S*i* SR*j*
001405	CCI	073*i*05	SR0 S*i*

IMI-mode Instructions

IMI mode is active when the monitor mode (MM) bit in the exchange package modes field is clear and the IMI bit in the exchange package interrupt modes field is set.

IMI mode is a special operating mode designed to facilitate testing of an operating system in a nondedicated CPU. The operating system being tested is run under the control of a supervisory program that runs in monitor mode. The test operating system runs in IMI mode.

The test operating system can run most instructions at full speed. However, monitor-mode instructions and instructions that affect the system environment or the environment of the test operating system are trapped. (Most trapped instructions execute as no-ops, but some execute normally.) After execution of a trapped instruction, an MII occurs. The supervisory program can then simulate the operation of the trapped instruction.

For proper operation, the cluster number (CLN) must be set to 0 when the CPU is operating in IMI mode. Table 4 lists all instructions that are trapped in IMI mode.

Table 4. IMI-mode Instructions

Machine Instruction	CAL Syntax	Operation When IMI Mode Active
0010*jk* (*jk* ≠ 0) 0011*jk* 0012*j*0	CA,A*j* A*k* CL,A*j* A*k* CI,A*j*	These instructions are privileged to monitor mode. They execute as no-ops in IMI mode. An MII interrupt occurs after the instruction executes.
0012*j*1	MC,A*j*	
0012*j*2	DI,A*j*	
0012*j*3	EI,A*j*	
0013*j*0	XA A*j*	
0013*j*1	A*j* XA	
001302	EMI	
001303	DMI	
0014*j*0	RT S*j*	
0014*j*1	SIPI A*j*	
001402 0014*j*3 0014*j*4	CIPI CLN A*j* PCI S*j*	
001405	CCI	
001406	ECI	
001407	DCI	
001500	—	

Machine Instruction	CAL Syntax	Operation When IMI Mode Active
001501	—	These instructions are privileged to monitor mode. They execute as no-ops in IMI mode. An MII interrupt occurs after the instruction executes.
001600	ESI	
001640	BCD	
0017*jk*	BP*k* A*j*	
023*ij*6	A*i* EA,*j*	
023*ij*7	A*i* EA,A*j*	
027*ij*2	EA*j* A*i*	
027*ij*3	EA,A*j* A*i*	
073*i*05	SR0 S*i*	
00200*k*	VL A*k*	These instructions execute normally in IMI mode. An MII interrupt occurs after the instruction executes.
072*i*00	S*i* RT	
073*i*01	S*i* SR0	
073*i*25 (no-op when in maintenance mode)	SR2 S*i*	
002100	EFI	These instructions execute normally in normal user mode, but execute as no-ops in IMI mode. An MII interrupt occurs after the instruction executes.
002200	DFI	
002210	CBL	
002300	ERI	
002301	EBP	
002400	DRI	
002401	DBP	
002500	DBM	
002501	ESC	
002600	EBM	
002601	DSC	
0034*jk* (*j*2 = 0)	SM*jk* 1,TS	Because the cluster number must be set to 0 when IMI mode is active, these instructions execute as no-ops. An MII interrupt occurs after the instruction executes.
0034*jk* (*j*2 = 1)	SM,A*k* 1,TS	
0036*jk* (*j*2 = 0)	SM*jk* 0	
0036*jk* (*j*2 = 1)	SM,A*k* 0	
0037*jk* (*j*2 = 0)	SM*jk* 1	
0037*jk* (*j*2 = 1)	SM,A*k* 1	
027*ij*6	SB,A*j* A*i*	
027*ij*7	SB*j* A*i*	
073*i*02	SM S*i*	
073*ij*3	ST*j* S*i*	
073*ij*6	ST,A*j* S*i*	
0064*jknm* (*j*2 = 0)	JTS*jk* *exp*	Because the cluster number must be set to 0 when IMI mode is active, these instructions execute as no-ops. An MII interrupt occurs after the instruction executes. Following the interrupt, the P register points to the second parcel (*m* field) of the instruction.
0064*jknm* (*j*2 = 1)	JTS,A*k* *exp*	
026*ij*4	A*i* SB,A*j*,+1	These instructions execute normally when IMI mode is active, but the data is blocked from entering register A*i*/S*i*. In addition, because the cluster number must be set to 0 when IMI mode is active, instructions 026*ij*4 and 026*ij*5 do not increment an SB register. An MII interrupt occurs after the instruction executes.
026*ij*5	A*i* SB*j*,+1	
026*ij*6	A*i* SB,A*j*	
026*ij*7	A*i* SB*j*	
072*i*02	S*i* SM	
072*ij*3	S*i* ST*j*	
072*ij*6	S*i* ST,A*j*	

Machine Instruction	CAL Syntax		Operation When IMI Mode Active
033*i*00 033*ij*0 (*j* ≠ 0) 033*ij*1 (*j* ≠ 0)	A*i* A*i* A*i*	CI CA,A*j* CE,A*j*	These instructions execute normally when IMI mode is active, but the data is blocked from entering register A*i*. This effectively makes them no-ops. An MII interrupt occurs after the instruction executes.
073*ij*1 (*j* = 2,3)	S*i*	SR*j*	This instruction is privileged to monitor mode. It executes normally in IMI mode except that the performance monitor pointer is prevented from advancing. An MII interrupt occurs after the instruction executes.
073*ij*1 (*j* = 4 – 7)	S*i*	SR*j*	This instruction is privileged to monitor mode. It clears register S*i* to 0 in IMI mode. An MII interrupt occurs after the instruction executes.
073*i*75	SR7	S*i*	This instruction operates as a no-op unless maintenance mode is active. With maintenance mode active, this instruction operates normally. An MII interrupt occurs after the instruction executes. **NOTE**: Normal use of this instruction requires checking of register SR0 bit 0 before executing the instruction. Because the instruction that does the checking (073*i*01) is trapped in IMI mode, it is recommended that instruction 073*i*75 not be used in IMI mode.

Instruction and Branch Timing

The instruction buffer attempts to keep ahead of instruction issue; this reduces instruction waiting times. Because the instruction set is complex and is executing in a complex environment, issue timing might not seem deterministic (due to things such as variable wait times for memory conflicts). However, some general rules can be stated for events that occur within a CPU.

Issue Timing

Although the instruction word that is the destination of a branch request is the first word requested from memory (followed by the remainder of the instruction block in circular order) instruction words can enter the stack in any order. (Eight words at a time are requested so that the 32-word block is requested over 4 clock periods.) Priority conflicts, however, can lengthen the request time.

The issue logic has five valid flags. The first flag corresponds to the branch address word. The next three flags correspond to the following 3 words (unless the branch address is 3 words or less from the end of a 32-word address block). The last flag indicates the validity of the remainder of the address block.

When the first valid flag sets, the issue unit retrieves the corresponding word from the buffer and starts issuing instructions. At the time the first parcel is issued, a request for the next word is made. The issue unit can request a new instruction word every 4 clock periods (CPs), corresponding to the maximum issue rate. The maximum issue rate is four 1-parcel instructions with no dependencies issued in 4 clock periods.

Issue continues until the next instruction word is required. If the next instruction word is available, issue continues; if the next word is not available, issue halts after the last complete instruction. (Instructions split across word boundaries are never issued until all parcels are available to the issue unit.) This sequence continues for the first four instruction words/valid flags.

Because the fifth valid flag indicates the validity of the remaining 28 words of the instruction block, issue halts after 4 instruction words unless the entire instruction block is available. This is true even if the first instruction issued is in the middle of the instruction block, with one exception. If the next sequential instruction word of the block enters the buffer in the same clock period that issue would halt, that word is sent to the issue unit without waiting.

In order to reduce delays caused by memory access times, a prefetch of the next sequential 32-word instruction block is requested when the 25th word (8th word from the end) of the current instruction block is entered or when a branch is done into the last 8 words of the current block. If the next instruction block is already in the buffer, it does not have to be fetched from memory. If the current block is still being fetched when the request for the next block occurs, the next block is not fetched until the current fetch is completed; the hardware can perform only one instruction fetch at a time.

A delay occurs if the first word of the next sequential instruction block is needed while the current block is still being fetched. In this case, issue halts after the last word of the first block until the first word of the next block is fetched.

If an out-of-stack branch occurs while the next sequential block is waiting to be prefetched, the prefetch is aborted and the block containing the branch address is fetched instead. Issue of instructions at the branch address are delayed until the fetch of the current block is completed, a fetch of the block containing the branch address can begin, and the requested instruction word is available from the instruction buffer.

If an in-stack branch occurs (either to the current block or to another block in the buffer) while the next sequential block is waiting to be prefetched, the prefetch is aborted. Because the word at the branch address is already in the buffer, no fetch is needed and issue continues without delay.

Branch Timing

In issuing, just like other instructions, a branch instruction is affected by instruction buffer timing and issue interlocks. In addition, timing is affected by branch success and by the destination address of the branch. Even if the destination address is currently in the instruction stack, timing is further affected by, for example, the destination parcel address and by the size (number of parcels) of the destination instruction.

Two timing numbers are given for branches: issue time and branch time. The issue time corresponds to the number of parcels in the instruction; most branch instructions are 3 parcels long and therefore take 3 clock periods to issue. The branch time listed is the minimum additional time required to complete an in-stack branch.

Branch fall-through, for conditional branches, requires no additional time. If a branch that is taken completes in 10 clock periods (3 CPs to issue and 7 CPs branch time) the fall-through time for that instruction is 3 CPs.

To the times listed, add additional time according to the rules in the following list. This time is in addition to the time required for out-of-stack instruction issues discussed previously and applies only to branches that are taken.

- If the destination parcel is parcel 0, no additional time is added.
- If the destination parcel is parcel 1 and the destination instruction is a 4-parcel instruction, add 1 CP to the branch time. (If it is not a 4-parcel instruction, do not add any time.)
- If the destination is parcel 2 and that instruction is a single parcel, add 1 CP. If it is a multiparcel instruction, add 2 CPs.
- If the destination parcel is parcel 3, add 2 CPs.

This timing can create a special case. If a branch to a multiparcel instruction in parcel 2 can be converted from a branch to a single parcel instruction in parcel 1 (even an inserted no-op before the multiparcel instruction), a CP can be saved even if the multiparcel instruction is not moved. (What would have been a 2-CP wait is converted to 1 CP to issue the single-parcel instruction.) If a 3-parcel instruction can be moved from parcel 2 to parcel 1, two CPs are saved.

Special CAL Syntax Forms

Certain machine instructions can be generated from two or more different CAL instructions. Any of the operations performed by special instructions can be performed by instructions in the basic CAL instruction set. For example, the following CAL instructions generate instruction 002000, which transmits a 1 to the vector length (VL) register:

- VL A0 (normal CAL syntax)
- VL 1 (special CAL syntax)

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

In several cases, a single CAL syntax can generate several different machine instructions. These cases provide for transmitting the value of an expression to an A register or S register, or for shifting A register or S register contents. For example, the CAL instruction A*i exp* generates instruction 020, 021, or 022, depending on the value of *exp.* The assembler uses *exp* to determine which instruction to generate.

Instruction Summary

Table 5 lists the special indicators that apply to many of the instructions. When one or more of these indicators applies to a specific instruction, the indicator is shown as a superscript letter following the machine instruction.

Table 6 lists, in numerical order, all instructions in the CRAY T90 series instruction set. Included for each instruction is the machine instruction, the CAL syntax, and a brief description.

Table 5. Special Indicators

Superscript	Description
N	New instruction (not available on CRAY C90 series systems)
V	New version of CRAY C90 series instruction
T	Triton mode only
D	Difference in operation between Triton mode and C90 mode
M	Monitor mode only
O	Maintenance mode only

Table 6. Instruction Special Indicators

Machine Instruction	CAL Syntax	Description
000000	ERR	Error exit.
001000	PASS	Pass (no operation).
0010*jk* (*jk* ≠ 0)^{M}	CA,A*j* A*k*	Set channel (A*j*) CA register (A*k*) and activate channel.
0011*jk*^{M}	CL,A*j* A*k*	Set channel (A*j*) CL register (A*k*).
0012*j*0^{M}	CI,A*j*	Clear interrupt flag and error flag for channel (A*j*). Clear Device Master Clear (output channels only). Enable channel interrupt.
0012*j*1^{M}	MC,A*j*	Clear interrupt flag and error flags for channel (A*j*). Set Device Master Clear (output channels only). Clear Ready Held (input channels only). Enable channel interrupt.
0012*j*2^{M}	DI,A*j*	Disable channel A*j* interrupt.
0012*j*3^{M}	EI,A*j*	Enable channel A*j* interrupt.
0013*j*0^{M}	XA A*j*	Transmit (A*j*) to exchange address.
0013*j*1^{NM}	A*j* XA	Transmit exchange address to A*j*.
001302^{M}	EMI	Enable monitor interrupt mode (set EIM to 1).
001303^{M}	DMI	Disable monitor interrupt mode (clear EIM to 0).
0014*j*0^{M}	RT S*j*	Transmit (S*j*) to real-time clock.
0014*j*1^{M}	SIPI A*j*	Send inter-CPU interrupt to CPU (A*j*).
001402^{M}	CIPI	Clear inter-CPU interrupt.
0014*j*3^{M}	CLN A*j*	Transmit (A*j*) to cluster number register.
0014*j*4^{M}	PCI S*j*	Transmit (S*j*) to programmable clock.
001405^{M}	CCI	Clear programmable clock interrupt (clear PCI to 0).

Machine Instruction	CAL Syntax	Description
001406^{M}	ECI	Enable programmable clock interrupt (set IPC to 1).
001407^{M}	DCI	Disable programmable clock interrupt (clear IPC to 0).
001500^{M}	—	Clear all performance monitor counters.
001501^{NTM}	—	Clear performance monitor pointer.
001600^{M}	ESI	Enable system I/O interrupts (set SIE to 1).
001640^{NM}	BCD	Broadcast cluster detach.
0017*jk*^{M}	BP,*k* A*j*	Transmit (A*j*) to breakpoint address *k* (*k* = 0 or 1).
00200*k*	VL A*k*	Transmit (A*k*) to vector length register.
002100	EFI	Enable interrupt on floating-point error (set IFP to 1).
002200	DFI	Disable interrupt on floating-point error (clear IFP to 0).
002210	CBL	Clear bit matrix loaded bit (clear BML to 0).
002300	ERI	Enable interrupt on operand range error (set IOR to 1).
002301	EBP	Enable interrupt on breakpoint (set IBP to 1).
002400	DRI	Disable interrupt on operand range error (clear IOR to 0).
002401	DBP	Disable interrupt on breakpoint (clear IBP to 0).
002500	DBM	Disable bidirectional memory transfers (clear BDM to 0).
002501^{N}	ESC	Enable scalar cache (set SCE to 1).
002600	EBM	Enable bidirectional memory transfers (set BDM to 1).
002601^{N}	DSC	Disable and invalidate scalar cache (clear SCE to 0).
002700	CMR	Complete memory references.
002704	CPA	Complete port reads and writes (ports A, B, and C).
002705	CPR	Complete port reads (ports A and B).
002706	CPW	Complete port writes (port C).
0030*j*0	VM0 S*j*	Transmit (S*j*) to VM0.
0030*j*1	VM1 S*j*	Transmit (S*j*) to VM1.
0030*j*2^{NT}	VM0 A*j*	Transmit (A*j*) to VM0.
0030*j*3^{NT}	VM1 A*j*	Transmit (A*j*) to VM1.
0034*jk* (*j*2 = 0)	SM*jk* 1,TS	Test and set semaphore *jk* (*jk* = 0 – 37$_8$).
0034*jk* (*j*2 = 1)	SM,A*k* 1,TS	Test and set semaphore (A*k*).
0036*jk* (*j*2 = 0)	SM*jk* 0	Clear semaphore *jk* (*jk* = 0 – 37$_8$).
0036*jk* (*j*2 = 1)	SM,A*k* 0	Clear semaphore (A*k*).

Machine Instruction	CAL Syntax	Description
0037*jk* (*j*2 = 0)	SM*jk* 1	Set semaphore *jk* (*jk* = 0 – 37_8).
0037*jk* (*j*2 = 1)	SM,A*k* 1	Set semaphore (A*k*).
00400*k*$^{\text{V}}$	EX*k*	Exit *k*.
0050*jk*	J B*jk*	Jump to B*jk*.
0051*jk*$^{\text{O}}$	JINV B*jk*	Jump to B*jk* (invalidate instruction buffers).
006000*nm*	J *exp*	Jump to *exp*.
006100*nm*$^{\text{NT}}$	IJ *exp*	Jump to address in *exp*.
0064*jknm* (*j*2 = 0)	JTS*jk* *exp*	Jump to *exp* if SM*jk* = 1; else set SM*jk*.
0064*jknm* (*j*2 = 1)	JTS,A*k* *exp*	Jump to *exp* if SM(A*k*) = 1; else set SM(A*k*).
007000*nm*	R *exp*	Return jump to *exp*; set B00 to (P)+3.
007100*nm*$^{\text{NT}}$	IR *exp*	Return jump to address in *exp*; set B00 to (P)+3.
010000*nm*$^{\text{D}}$	JAZ *exp*	Jump to *exp* if (A0) = 0.
011000*nm*$^{\text{D}}$	JAN *exp*	Jump to *exp* if (A0) ≠ 0.
012000*nm*$^{\text{D}}$	JAP *exp*	Jump to *exp* if (A0) ≥ 0.
013000*nm*$^{\text{D}}$	JAM *exp*	Jump to *exp* if (A0) < 0.
014000*nm*	JSZ *exp*	Jump to *exp* if (S0) = 0.
015000*nm*	JSN *exp*	Jump to *exp* if (S0) ≠ 0.
016000*nm*	JSP *exp*	Jump to *exp* if (S0) ≥ 0.
017000*nm*	JSM *exp*	Jump to *exp* if (S0) < 0.
020*i*00*nm*$^{\text{D}}$	A*i* *exp*	Transmit *nm* to A*i* bits 0 – 31; A*i* bits 32 – 63 = 0.
020*i*20*nm*$^{\text{NT}}$	A*i* A*i*:*exp*	Transmit *nm* to A*i* bits 0 – 31; A*i* bits 32 – 63 unchanged.
020*i*40*nm*$^{\text{NT}}$	A*i* *exp*:A*i*	Transmit *nm* to A*i* bits 32 – 63; A*i* bits 0 – 31 unchanged.
021*i*00*nm*$^{\text{D}}$	A*i* *exp*	Transmit inverse (*nm*) to A*i* bits 0 – 31; A*i* bits 32 – 63 = 1.
022*ijk*	A*i* *exp*	Transmit *jk* to A*i* bits 0 – 5; A*i* bits 6 – 63 = 0.
023*ij*0$^{\text{D}}$	A*i* S*j*	Transmit (S*j*) to A*i*.
023*i*01	A*i* VL	Transmit (VL) to A*i*.
023*ij*6$^{\text{NM}}$	A*i* EA,*j*	Transmit exit address *j* to A*i*.
023*ij*7$^{\text{NM}}$	A*i* EA,A*j*	Transmit exit address (A*j*) to A*i*.
024*ijk*$^{\text{D}}$	A*i* B*jk*	Transmit (B*jk*) to A*i*.
025*ijk*$^{\text{D}}$	B*j* A*i*	Transmit (A*i*) to B*jk*.
026*ij*0	A*i* PS*j*	Transmit population count of (S*j*) to A*i*.
026*ij*1	A*i* QS*j*	Transmit population count parity of (S*j*) to A*i*.
026*ij*2$^{\text{ND}}$	A*i* PA*j*	Transmit population count of (A*j*) to A*i*.
026*ij*3$^{\text{ND}}$	A*i* QA*j*	Transmit population count parity of (A*j*) to A*i*.
026*ij*4$^{\text{D}}$	A*i* SB,A*j*,+1	Transmit (SB(A*j*)) to A*i*; increment (SB(A*j*)) by 1.
026*ij*5$^{\text{D}}$	A*i* SB*j*,+1	Transmit (SB*j*) to A*i*; increment (SB*j*) by 1.

Machine Instruction	CAL Syntax		Description
026*ij*6$^{\text{D}}$	A*i*	SB,A*j*	Transmit (SB(A*j*)) to A*i*.
026*ij*7$^{\text{D}}$	A*i*	SB*j*	Transmit (SB*j*) to A*i*.
027*ij*0	A*i*	ZS*j*	Transmit leading zero count of (S*j*) to A*i*.
027*ij*1$^{\text{NT}}$	A*i*	ZA*j*	Transmit leading zero count of (A*j*) to A*i*.
027*ij*2$^{\text{NM}}$	EA*j*	A*i*	Transmit (A*i*) to exit address *j*.
027*ij*3$^{\text{NM}}$	EA,A*j*	A*i*	Transmit (A*i*) to exit address (A*j*).
027*ij*6$^{\text{D}}$	SB,A*j*	A*i*	Transmit (A*i*) to SB(A*j*).
027*ij*7$^{\text{D}}$	SB*j*	A*i*	Transmit (A*i*) to SB*j*.
030*ijk*$^{\text{D}}$	A*i*	A*j*+A*k*	Transmit integer sum of (A*j*) and (A*k*) to A*i*.
031*ijk*$^{\text{D}}$	A*i*	A*j*-A*k*	Transmit integer difference (A*j*) and (A*k*) to A*i*.
032*ijk*$^{\text{D}}$	A*i*	A*j**A*k*	Address multiply.
033*i*00$^{\text{DM}}$	A*i*	CI	Transmit channel number of highest-priority interrupt request to A*i*.
033*ij*0 ($j \neq 0$)$^{\text{DM}}$	A*i*	CA,A*j*	Transmit current address of channel (A*j*) to register A*i*.
033*ij*1 ($j \neq 0$)$^{\text{DM}}$	A*i*	CE,A*j*	Transmit status/error word of channel (A*j*) to register A*i*.
034*ijk*$^{\text{D}}$	B*jk*,A*i*	,A0	Transmit (A*i*) words from common memory starting at address (A0) to B registers starting at register *jk*.
035*ijk*$^{\text{D}}$	,A0	B*jk*,A*i*	Transmit (A*i*) words from B registers starting at register *jk* to memory starting at address (A0).
036*ijk*$^{\text{D}}$	T*jk*,A*i*	,A0	Transmit (A*i*) words from memory starting at address (A0) to T registers starting at register *jk*.
037*ijk*$^{\text{D}}$	,A0	T*jk*,A*i*	Transmit (A*i*) words from T registers starting at register *jk* to memory starting at address (A0).
040*i*00*nm*	S*i*	*exp*	Transmit *nm* to S*i* bits – 31; S*i* bits 32 – 63 = 0.
040*i*20*nm*	S*i*	S*i*:*exp*	Transmit *nm* to S*i* bits 0 – 31; S*i* bits 32 – 63 unchanged.
040*i*40*nm*	S*i*	*exp*:S*i*	Transmit *nm* to S*i* bits 32 – 63; S*i* bits 0 – 31 unchanged.
041*i*00*nm*	S*i*	*exp*	Transmit inverse (*nm*) to S*i* bits 0 – 31; S*i* bits 32 – 63 = 1.
042*ijk*	S*i*	<*exp*	Form ones mask in S*i* *exp* bits from right; *jk* field gets 100_8 – *exp*.
005400 042*ijk*$^{\text{NT}}$	A*i*	<*exp*	Form ones mask in A*i* *exp* bits from right; *jk* field gets 100_8 – *exp*.
043*ijk*	S*i*	>*exp*	Form ones mask in S*i* *exp* bits from left; *jk* field gets *exp*.
005400 043*ijk*$^{\text{NT}}$	A*i*	>*exp*	Form ones mask in A*i* *exp* bits from left; *jk* field gets *exp*.
044*ijk*	S*i*	S*j*&S*k*	Transmit logical product of (S*j*) and (S*k*) to S*i*.

Machine Instruction	CAL Syntax	Description
005400 044*ijk*^{NT}	A*i* A*j*&A*k*	Transmit logical product of (A*j*) and (A*k*) to A*i*.
045*ijk*	S*i* #S*k*&S*j*	Transmit logical product of (S*j*) and one's complement of (S*k*) to S*i*.
005400 045*ijk*^{NT}	A*i* #A*k*&A*j*	Transmit logical product of (A*j*) and one's complement of (A*k*) to A*i*.
046*ijk*	S*i* S*j*\S*k*	Transmit logical difference of (S*j*) and (S*k*) to S*i*.
005400 046*ijk*^{NT}	A*i* A*j*\A*k*	Transmit logical difference of (A*j*) and (A*k*) to A*i*.
047*ijk*	S*i* #S*j*\S*k*	Transmit logical equivalence of (S*j*) and (S*k*) to S*i*.
005400 047*ijk*^{NT}	A*i* #A*j*\A*k*	Transmit logical equivalence of (A*j*) and (A*k*) to A*i*.
050*ijk*	S*i* S*j*!S*i*&S*k*	Merge (S*i*) and (S*j*) to S*i* using (S*k*) as mask.
005400 050*ijk*^{NT}	A*i* A*j*!A*i*&A*k*	Merge A*i* and A*j* to A*i* using (A*k*) as mask.
051*ijk*	S*i* S*j*!S*k*	Transmit logical sum of (S*j*) and (S*k*) to S*i*.
005400 051*ijk*^{NT}	A*i* A*j*!A*k*	Transmit logical sum of (A*j*) and (A*k*) to A*i*.
052*ijk*	S0 S*i*<*exp*	Shift (S*i*) left *exp* = *jk* places to S0.
005400 052*ijk*^{NT}	A0 A*i*<*exp*	Shift (A*i*) left *exp* = *jk* places to A0.
053*ijk*	S0 S*i*>*exp*	Shift (S*i*) right *exp* = 100_8 – *jk* places to S0.
005400 053*ijk*^{NT}	A0 A*i*>*exp*	Shift (A*i*) right *exp* = 100_8 – *jk* places to A0.
054*ijk*	S*i* S*i*<*exp*	Shift (S*i*) left *exp* = *jk* places to S*i*.
005400 054*ijk*^{NT}	A*i* A*i*<*exp*	Shift (A*i*) left *exp* = *jk* places to A*i*.
055*ijk*	S*i* S*i*>*exp*	Shift (S*i*) right *exp* = 100_8 – *jk* places to S*i*.
005400 055*ijk*^{NT}	A*i* A*i*>*exp*	Shift (A*i*) right *exp* = 100_8 – *jk* places to A*i*.
056*ijk*^{D}	S*i* S*i*,S*j*<A*k*	Shift (S*i*) and (S*j*) left (A*k*) places to S*i*.
005400 056*ijk*^{NT}	A*i* A*i*,A*j*<A*k*	Shift (A*i*) and (A*j*) left (A*k*) places to A*i*.
057*ijk*^{D}	S*i* S*j*,S*i*>A*k*	Shift (S*j*) and (S*i*) right (A*k*) places to S*i*.
005400 057*ijk*^{NT}	A*i* A*j*,A*i*>A*k*	Shift (A*j*) and (A*i*) right (A*k*) places to A*i*.
060*ijk*	S*i* S*j*+S*k*	Transmit integer sum of (S*j*) and (S*k*) to S*i*.
061*ijk*	S*i* S*j*-S*k*	Transmit integer difference of (S*j*) and (S*k*) to S*i*.
062*ijk*	S*i* S*j*+FS*k*	Transmit floating-point sum of (S*j*) and (S*k*) to S*i*.
063*ijk*	S*i* S*j*-FS*k*	Transmit floating-point difference of (S*j*) and (S*k*) to S*i*.
064*ijk*	S*i* Sj*FSk	Transmit floating-point product of (S*j*) and (S*k*) to S*i*.
065*ijk*	S*i* Sj*HSk	Transmit half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*.
066*ijk*	S*i* Sj*RSk	Transmit rounded floating-point product of (S*j*) and (S*k*) to S*i*.
067*ijk*	S*i* Sj*ISk	Transmit 2 – (S*j*) * (S*k*) to S*i* (reciprocal iteration).
070*ij*0	S*i* /HS*j*	Transmit floating-point reciprocal approximation of (S*j*) to S*i*.

Machine Instruction	CAL Syntax		Description
070*ij*1$^{\mathrm{N}}$	V*i*	CI,S*j*&VM	Transmit compressed index of (S*j*) controlled by (VM) to V*i*.
070*ij*6$^{\mathrm{N}}$	S*i*	S*j**BT	Transmit bit-matrix product of (S*j*) and (B$^{\mathrm{T}}$) to S*i*.
071*i*0*k*$^{\mathrm{D}}$	S*i*	A*k*	Transmit (A*k*) with no sign extension to S*i*.
071*i*1*k*$^{\mathrm{D}}$	S*i*	+A*k*	Transmit (A*k*) with sign extension to S*i* .
071*i*2*k*$^{\mathrm{D}}$	S*i*	+FA*k*	Transmit (A*k*) as unnormalized floating-point number to S*i*.
071*i*30	S*i*	0.6	Transmit 0.75 x 2^{48} as normalized floating-point constant to S*i*.
071*i*40	S*i*	0.4	Transmit 0.4_8 as normalized floating-point constant to S*i*.
071*i*50	S*i*	1.0	Transmit 1.0 as normalized floating-point constant to S*i*.
071*i*60	S*i*	2.0	Transmit 2.0 as normalized floating-point constant to S*i*.
071*i*70	S*i*	4.0	Transmit 4.0 as normalized floating-point constant to S*i*.
072*i*00	S*i*	RT	Transmit real-time clock to S*i*.
072*i*02$^{\mathrm{V}}$	S*i*	SM	Transmit semaphores to S*i*.
072*ij*3	S*i*	ST*j*	Transmit (ST*j*) register to S*i*.
072*ij*6$^{\mathrm{V}}$	S*i*	ST,A*j*	Transmit ST(A*j*) to S*i*.
073*i*00	S*i*	VM0	Transmit (VM0) to S*i*.
073*i*10	S*i*	VM1	Transmit (VM1) to S*i*.
073*i*20$^{\mathrm{NT}}$	A*i*	VM0	Transmit (VM0) to A*i*.
073*i*30$^{\mathrm{NT}}$	A*i*	VM1	Transmit (VM1) to A*i*.
073*ij*1$^{\mathrm{VM}}$	S*i*	SR*j*	Transmit (SR*j*) to S*i* (monitor mode only for *j* = 2 – 7).
073*i*02$^{\mathrm{V}}$	SM	S*i*	Transmit (S*i*) to semaphores.
073*ij*3	ST*j*	S*i*	Transmit (S*i*) to ST*j*.
073*i*05	SR0	S*i*	Transmit (S*i*) bits 48 – 52 to SR0.
073*i*25$^{\mathrm{O}}$	SR2	S*i*	Advance performance monitor pointer.
073*i*75$^{\mathrm{VO}}$	SR7	S*i*	Transmit (S*i*) to maintenance channel.
073*ij*6$^{\mathrm{V}}$	ST,A*j*	S*i*	Transmit (S*i*) to ST (A*j*).
074*ijk*	S*i*	T*jk*	Transmit (T*jk*) to S*i*.
075*ijk*	T*jk*	S*i*	Transmit (S*i*) to T*jk*.
076*ijk*	S*i*	*Vj*,A*k*	Transmit (V*j* element (A*k*)) to S*i*.
077*ijk*	V*i*,A*k*	S*j*	Transmit (S*j*) to V*i* element (A*k*).
10*hi*00*nm*$^{\mathrm{D}}$	A*i*	*exp*,A*h*	Load A*i* from ((A*h*) + *exp*).
10*hi*20*nm*$^{\mathrm{ND}}$	A*i*	*exp*,A*h*,BC	Load A*i* from ((A*h*) + *exp*) bypassing data cache and invalidating cache line.
10*hi*40*pnm*$^{\mathrm{NT}}$	A*i*	*exp*,A*h*	Load A*i* from ((A*h*) + *exp*).

Machine Instruction	CAL Syntax	Description
10*hi*60*pnm*^{NT}	A*i* *exp*,A*h*,BC	Load A*i* from ((A*h*) + *exp*) bypassing data cache and invalidating cache line.
11*hi*00*nm*^{D}	*exp*,A*h* A*i*	Store (A*i*) to ((A*h*) + *exp*).
11*hi*40*pnm*^{NT}	*exp*,A*h* A*i*	Store (A*i*) to ((A*h*) + *exp*).
12*hi*00*nm*	S*i* *exp*,A*h*	Load S*i* from ((A*h*) + *exp*).
12*hi*20*nm*^{N}	S*i* *exp*,A*h*,BC	Load S*i* from ((A*h*) + *exp*) bypassing data cache and invalidating cache line.
12*hi*40*pnm*^{NT}	S*i* *exp*,A*h*	Load S*i* from ((A*h*) + *exp*).
12*hi*60*pnm*^{NT}	S*i* *exp*,A*h*,BC	Load S*i* from ((A*h*) + *exp*) bypassing data cache and invalidating cache line.
13*hi*00*nm*	*exp*,A*h* S*i*	Store (S*i*) to ((A*h*) + *exp*).
13*hi*40*pnm*^{NT}	*exp*,A*h* S*i*	Store (S*i*) to ((A*h*) + *exp*).
140*ijk*	V*i* S*j*&V*k*	Transmit logical products of (S*j*) and (V*k* elements) to V*i* elements.
141*ijk*	V*i* V*j*&V*k*	Transmit logical products of (V*j* elements) and (V*k* elements) to V*i* elements.
142*ijk*	V*i* S*j*!V*k*	Transmit logical sums of (S*j*) and (V*k* elements) to V*i* elements.
143*ijk*	V*i* V*j*!V*k*	Transmit logical sums of (V*j* elements) and (V*k* elements) to V*i* elements.
144*ijk*	V*i* S*j*\V*k*	Transmit logical differences of (S*j*) and (V*k* elements) to V*i* elements.
145*ijk*	V*i* V*j*\V*k*	Transmit logical differences of (V*j* elements) and (V*k* elements) to V*i* elements.
146*ijk*	V*i* S*j*!V*k*&VM	Merge (S*j*) and (V*k* elements) to V*i* elements using (VM) as mask.
147*ijk*	V*i* V*j*!V*k*&VM	Merge (V*j* elements) and (V*k* elements) to V*i* elements using (VM) as mask.
150*ijk*^{D}	V*i* V*j*<A*k*	Shift (V*j* elements) left (A*k*) places to V*i* elements.
005400 150*ij*0	V*i* V*j*<V0	Shift (V*j* elements) left (V0 elements) places to V*i* elements.
151*ijk*^{D}	V*i* V*j*>A*k*	Shift (V*j* elements) right (A*k*) places to V*i* elements.
005400 151*ij*0	V*i* V*j*>V0	Shift (V*j* elements) right (V0 elements) places to V*i* elements.
152*ijk*	V*i* V*j*,V*j*<A*k*	Double shift (V*j* elements) left (A*k*) places to V*i* elements.
005400 152*ijk*	V*i* V*j*,A*k*	Transfer (V*j* elements) starting at element (A*k*) to V*i* elements.
153*ijk*	V*i* V*j*,V*j*>A*k*	Double shift (V*j* elements) right (A*k*) places to V*i* elements.
005400 153*ij*0^{NT}	V*i* V*j*,[VM]	Compress V*j* by (VM) to V*i*.
005400 153*ij*1^{NT}	V*i*,[VM] V*j*	Expand V*j* by (VM) to V*i*.

Machine Instruction	CAL Syntax	Description
154ijk	Vi Sj+Vk	Transmit integer sums of (Sj) and (Vk elements) to Vi elements.
155ijk	Vi Vj+Vk	Transmit integer sums of (Vj elements) and (Vk elements) to Vi elements.
156ijk	Vi Sj-Vk	Transmit integer differences of (Sj) and (Vk elements) to Vi elements.
157ijk	Vi Vj-Vk	Transmit integer differences of (Vj elements) and (Vk elements) to Vi elements.
160ijk	Vi Sj*FVk	Transmit floating-point products of (Sj) and (Vk elements) to Vi elements.
161ijk	Vi Vj*FVk	Transmit floating-point products of (Vj elements) and (Vk elements) to Vi elements.
162ijk	Vi Sj*HVk	Transmit half-precision rounded floating-point products of (Sj) and (Vk elements) to Vi elements.
163ijk	Vi Vj*HVk	Transmit half-precision rounded floating-point products of (Vj elements) and (Vk elements) to Vi elements.
164ijk	Vi Sj*RVk	Transmit rounded floating-point products of (Sj) and (Vk elements) to Vi elements.
165ijk	Vi Vj*RVk	Transmit rounded floating-point products of (Vj elements) and (Vk elements) to Vi elements.
166ijk^{D}	Vi Sj*Vk	Transmit integer products of (Sj) and (Vk elements) to Vi elements.
167ijk	Vi Vj*Vk	Transmit 2 – the integer products of (Vj elements) and (Vk elements) to Vi elements (reciprocal iteration).
170ijk	Vi Sj+FVk	Transmit floating-point sums of (Sj) and (Vk elements) to Vi elements.
171ijk	Vi Vj+FVk	Transmit floating-point sums of (Vj elements) and (Vk elements) to Vi elements.
172ijk	Vi Sj-FVk	Transmit floating-point differences of (Sj) and (Vk elements) to Vi elements.
173ijk	Vi Vj-FVk	Transmit floating-point differences of (Vj elements) and (Vk elements) to Vi elements.
174ij0	Vi /HVj	Transmit floating-point reciprocal approximation of (Vj elements) to Vi elements.
174ij1	Vi PVj	Transmit population count of (Vj elements) to Vi elements.
174ij2	Vi QVj	Transmit population count parity of (Vj elements) to Vi elements.
174ij3	Vi ZVj	Transmit leading zero count of (Vj elements) to Vi elements.
1740j4	BMM LVj	Transmit Vj elements 0 – 63 to B matrix.
1740j5^{N}	BMM UVj	Transmit Vj elements 64 – 127 to B matrix.

Machine Instruction	CAL Syntax		Description
174_ij_6	V_i_	V_j_*BT	Transmit bit-matrix product of (V_j_) and (B^T) to V_i_.
1750_j_0	VM	V_j_,Z	Set VM bit if (V_j_ element) = 0.
1750_j_1	VM	V_j_,N	Set VM bit if (V_j_ element) ≠ 0.
1750_j_2	VM	V_j_,P	Set VM bit if (V_j_ element) ≥ 0.
1750_j_3	VM	V_j_,M	Set VM bit if (V_j_ element) < 0.
175_ij_4	V_i_,VM	V_j_,Z	Set VM bit if (V_j_ element) = 0 and store compressed indices of V_j_ elements = 0 in V_i_.
175_ij_5	V_i_,VM	V_j_,N	Set VM bit if (V_j_ element) ≠ 0 and store compressed indices of V_j_ elements ≠ 0 in V_i_.
175_ij_6	V_i_,VM	V_j_,P	Set VM bit if (V_j_ element) ≥ 0 and store compressed indices of V_j_ elements ≥ 0 in V_i_.
175_ij_7	V_i_,VM	V_j_,M	Set VM bit if (V_j_ element) < 0 and store compressed indices of V_j_ elements < 0 in V_i_.
176_i_0_k_	V_i_	,A0,A_k_	Load V_i_ from memory starting at address (A0) and incrementing by (A_k_).
176_i_1_k_	V_i_	,A0,V_k_	Load V_i_ from memory using addresses (A0) + (V_k_).
005400 176_ijk_$^{\text{NT}}$	V_i_:V_j_	,A0:A_k_,V_k_	Load V_i_ from memory using addresses (A0) + (V_k_) and load V_j_ from memory using addresses (A_k_) + (V_k_).
1770_jk_	,A0,A_k_	V_j_	Store (V_j_) to memory starting at address (A0) and increment by (A_k_).
1771_jk_	,A0,V_k_	V_j_	Store (V_j_) to memory using addresses (A0) + (V_k_).