Chapter 4

Instruction Set

CHAPTER 4. INSTRUCTION SET

4 INSTRUCTION SET 4-1
Figure 4-1 Register Programming Model 4-2
4.1 Flags 4-3
Table 4-1 DSP3210 Flags 4-3
4.2 DA Instructions 4-4
4.2.1 DA Instructions - Multiply/Accumulate 4-4
Table 4-2 DA Multiply/Accumulate Instructions 4-4
Table 4-3 Replacement for DA Multiply/Accumulate Instructions 4-5
4.2.2 DA Instructions - Special Functions 4-5
Table 4-4 DA Special Function Instructions 4-5
Table 4-5 Replacement for DA Special Function Instructions 4-5
4.3 CA Instructions 4-6
4.3.1 CA Instructions - Control 4-6
Table 4-6 CA Control Instructions 4-6
Table 4-7 Replacement for COND 4-7
4.3.2 CA Instructions - Arithmetic/Logic 4-8
Table 4-8 CA Arithmetic/Logic Instructions 4-9
4.3.3 CA Instructions - Data Move 4-10
Table 4-9 CA Data Move Group Instructions 4-11
Table 4-10 Replacement for CA Data Move Instructions 4-11
4.4 DSP3210 Programming 4-12
4.4.1 Restrictions 4-12
4.4.1.1 Restriction 1 - Z-Field Pointer 4-12
4.4.1.2 Restriction 2 - CAU and IO Register Store 4-12
4.4.1.3 Restriction 3 - CAU Register Load 4-13
4.4.2 Latency Effects 4-13
4.4.2.1 Latency 1 - DA Instruction - Memory Writes 4-14
4.4.2.2 Latency 2 - Accumulator as Multiplier Input 4-14
4.4.2.3 Latency 3 - Branching 4-14
4.4.2.4 Latency 4 - Conditional Branching on DA Conditions 4-15
4.4.3 Common Programming Techniques 4-15
4.4.3.1 Subroutine Nesting 4-15
4.4.3.2 Passing Parameters 4-16
4.4.3.3 Quick Interrupt Facility 4-17
4.5 Common Signal Processing Functions 4-17
4.5.1 Standard Signal Processing Functions 4-17
4.5.2 Nonlinear Functions 4-18
4.6 Detailed Description of Instruction Set 4-20

4. INSTRUCTION SET

The DSP3210 has two general types of instructions that correspond to the two execution units: data arithmetic (DA) instructions and control arithmetic (CA) instructions. Primarily, DA instructions perform 32-bit floating-point multiply/accumulate operations for signal processing algorithms. Other DA instructions convert the DSP's internal floating-point data to and from each of the following types: 8-, 16-, or 32-bit integer, μ-law, A-law, or single-precision IEEE floating-point format. The CA instructions perform microprocessor operations such as 16- and 32-bit integer arithmetic and logic functions, conditional branching, and moving data. Instruction bit encodings are contained in Section 10 - Instruction and Register Set Encodings.

Figure 4-1 shows the register programming model of the DSP3210. It consists of twenty-two 32-bit general purpose registers, one register hardwired to zero, a 32-bit program counter, four 40-bit floating-point accumulator registers, and five IO control registers. These registers are summarized below:

- General-purpose CAU 32-bit registers, r1—r22
- CAU register hard-wired to zero, r0
- 32-bit program counter, pc, and associated interrupt shadow register, pcsh
- 40-bit floating-point accumulator registers, a0—a3, and associated interrupt shadow registers
- Control registers
  –Processor status register, ps, and associated shadow register
  –Exception mask register, emr
  –Processor control word, pcw
  –DAU control register, dauc, and associated shadow register
  –Clip-test register, ctr, and associated shadow register

Although r1—r22 can be used for general purposes, under certain circumstances these registers are designated to be used in specific ways:

Register	Name	Meaning
r1—r14	rP	DA instruction memory reference (X, Y, Z fields) pointer registers.
r15—r19	rI	DA instruction memory reference (X, Y, Z fields) increment registers.
r20	r20	Used by error exception facility to store the old pc.
r21	sp	Stack pointer.
r22	evtp	Used as the pointer to the exception vector table.

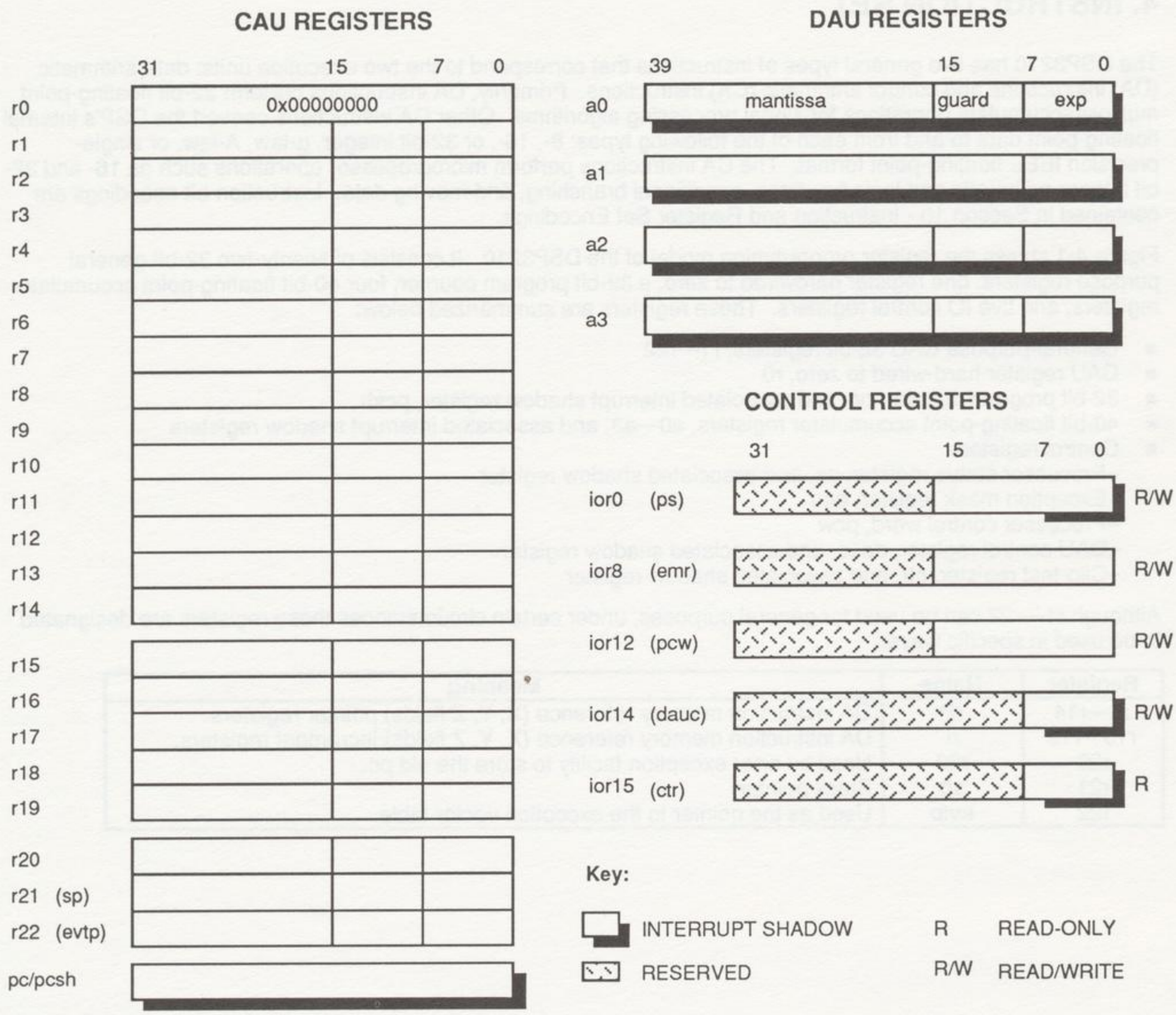

Figure 4-1 Register Programming Model

4.1 Flags

The DSP3210 has internal flags that are affected by the results of certain DA, or CA instructions and certain I/O events. These flags may be tested by conditional instructions and may also be read from the ps register. Table 4-1 lists the flags and their meaning.

DSP3210 instructions and the flags affected by each instruction are specified in sections 4.2 DA Instructions and 4.3 CA Instructions. A zero (0) shown in place of a flag means that the flag is always made zero; a dash (-) in place of a flag means that the flag is unaffected by the instruction.

Table 4-1 DSP3210 Flags

DAU FLAGS				
Flag	**True**	**False**	**Meaning (Flag = 1)**	**Definition**
N	alt	age	Result is negative	sign bit = one
Z	aeq	ane	Result is zero	mantissa = zero
V	avs	avc	Result overflowed	|DAU result | > 3.40282E38
U	aus	auc	Result underflowed	|DAU result | < 5.87747E-39

CAU FLAGS				
Flag	**True**	**False**	**Meaning (Flag = 1)**	**Definition**
n	mi	pl	Result is negative	(16-bit) n=b31 (bit 31 of ALU result) (16-bit) n=b15 (bit 15 of Barrel Shifter result) (32-bit) n=b31
z	eq	ne	Result is zero	(16-bit) z=b31 + b30 + ... b1 + b0 (+=OR) (32-bit) z=b31 + b30 + ... b1 + b0
c	cs	cc	Carry or borrow out of MSB	(16-bit) c=b15c (carry out of ALU bit 15) (32-bit) c=b31c
v	vs	vc	Result overflowed	(16-bit) v=b14c^b15c (^=exclusive OR) (32-bit) v=b30c^b31c

I/O FLAGS				
Flag	**True**	**False**	**Meaning (Flag = 1)**	**Definition**
i	ibf	ibe	Serial input buffer full	IBF = 1
o	obf	obe	Serial output buffer full	OBE = 0
s	sys	syc	SY (I/O sync) set	SY = 1
b	fbs	fbc	SIO frame boundary	First OBE after SY high-to-low transition
r	ir0s	ir0c	Interrupt 0 set	IR0N = 1
R	ir1s	ir1c	Interrupt 1 set	IR1N = 1

4.2 DA Instructions

The data arithmetic (DA) instructions are divided into two functional groups: multiply/accumulate and special funtions.

4.2.1 DA Instructions - Multiply/Accumulate

DA multiply/accumulate instructions perform 32-bit floating-point multiply/accumulate operations for signal processing algorithms. The DA multiply/accumulate instructions are listed in Table 4-2. Replacements for the operands are listed in Table 4-3.

Note: { } and [ ] are not part of the instruction syntax. Parentheses ( ) are part of the syntax and must appear where shown in an instruction. Lower-case letters are part of the syntax and upper-case letters are replaced by immediate data or by a register name (see tables following each instruction group). A detailed description of each instruction is available in Section 4.6 under the Instruction Reference name.

Table 4-2 DA Multiply/Accumulate Instructions

Instruction	DAU Flags	Instr. Reference	Description
[Z=] aN = [–]aM {+, –}Y*X	NZVU	FMULT-ACC-STORE	The product of the X and Y fields is added/subtracted to/from the accumulator aM and the result is stored in accumulator aN. The result can also be output according to the Z field.
aN = [–]aM {+, –}(Z=Y)*X	NZVU	FMULT-ACC-TAP	The Y field operand is output according to the Z field. The product of the X and Y fields is added to accumulator aM and the sum is stored in accumulator aN.
[Z=] aN = [–]Y {+, –}aM*X	NZVU	FMULT-ADD-STORE	The product of the X field and accumulator aM is added/subtracted to/from the Y field. The result is placed in accumulator aN and can also be output according to the Z field.
[Z=] aN = [–]Y *X	NZVU	FMULT-STORE	The product of the X and Y fields is added/subtracted to/from zero. The result is stored in accumulator aN and can also be output according to the Z field.
aN = [–] (Z=Y)*X	NZVU	FMULT-TAP	The value of the Y field is output according to the Z field. The product of the X and Y fields is stored in accumulator aN.
[Z=] aN = [–]Y {+, –} X	NZVU	FMULT-STORE	The sum or difference of the Y and X fields is stored in accumulator aN and the result can also be output according to the Z field.
[Z=] aN = [–]Y	NZVU	FLOAD-STORE	The value of the Y field is placed in accumulator aN and can also be output according to the Z field.
aN = [–] (Z=Y) {+, –} X	NZVU	FADD-TAP	The sum or difference of the X and Y fields is stored in accumulator aN, and Y is output according to the Z field.

Table 4-3 Replacement for DA Multiply/Accumulate Instructions

Replace	Value*	Meaning
aN, aM	a0-a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI,	32-bit memory location, or
	a0-a3	one of the four DAU accumulators
Z	*rP, *rP++, *rP--, *rP++rI	32-bit memory location

* rP refers to r1—r14 and is used as a memory pointer. rI refers to r15—r19 and is used as an increment register.

4.2.2 DA Instructions - Special Function

DA special function instructions perform data type format conversions, rounding, conditional accumulator loads, and compute a reciprocal seed. The DA special function instructions are listed in Table 4–4. Replacements for the operands are listed in Table 4-5.

Table 4–4 DA Special Function Instructions

Instruction	DAU Flags	Instr. Reference	Description
[Z=] aN = ic(Y)	NZ00	IC	Input conversion μ-law, A-law, 8-bit linear to float.
[Z=] aN = oc(Y)	—	OC	Output conversion float to μ-law, A-law, 8-bit linear.
[Z=] aN = float16(Y)	NZ00	FLOAT16	16-bit integer to float.
[Z=] aN = float32(Y)	NZ00	FLOAT32	32-bit integer to float.
[Z=] aN = int16(Y)	—	INT16	Float to 16-bit integer (round or truncate, dauc[4]).
[Z=] aN = int32(Y)	—	INT32	Float to 32-bit integer (round or truncate, dauc[4]).
[Z=] aN = round(Y)	NZVU	ROUND	Round to nearest, float(40) to float(32).
[Z=] aN = ifalt(Y)	—	IFALT	Conditional assignment/memory write.
[Z=] aN = ifaeq(Y)	—	IFAEQ	Conditional assignment/memory write.
[Z=] aN = ifagt(Y)	—	IFAGT	Conditional assignment/memory write.
[Z=] aN = dsp(Y)	NZVU	DSP	IEEE to DSP format conversion.
[Z=] aN = ieee(Y)	—	IEEE	DSP to IEEE format conversion.
[Z=] aN = seed(Y)	NZ0U	SEED	32-bit to 32-bit reciprocal seed.

Table 4-5 Replacement for DA Special Function Instructions

Replace	Value*	Meaning
aN	a0-a3	One of the four DAU accumulators
Y†	*rP, *rP++, *rP--, *rP++rI,	8-, 16-, or 32-bit memory location, or
	a0-a3	one of the four DAU accumulators
Z	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location

* rP refers to r1—r14 and is used as a memory pointer. rI refers to r15—r19 and is used as an increment register.
† Y may not be a0—a3 for the *dsp* special function.

4.3 CA Instructions

The control arithmetic (CA) instructions perform 16- and 32-bit integer operations. These operations include generating addresses, performing integer data and logical functions, instruction flow control, and data movement. The instructions are grouped into three major headings: control, arithmetic and logic, and data move.

4.3.1 CA Instructions - Control

The CA control instructions alter the ordinary flow of control from one instruction to the next. They include branches, loop counter test, subroutine call and return, return from interrupt, do loops, and wait for interrupt. Instructions can branch based on CA, DA, or I/O conditions. The condition for the branch is determined by the state of the flags resulting from the most recently calculated CA or DA instruction or I/O event. The CA control instructions have no effect on the flags. The CA control instructions are listed in Table 4-6. Replacements for the COND field are listed in Table 4-7.

Table 4-6 CA Control Instructions

Instruction	Instr. Reference	Description
if (COND) goto {N, rB, rB+N}*	GOTO-COND	Conditional branch based on flags
if (rM-->=0) goto {N, rB, rB+N}*	GOTO-LOOP	Conditional branch using loop counter
goto {N, rB, rB+N, M, rB+M}*	GOTO	Unconditional branch
nop	NOP	No operation
call {N, rB, rB+N, M} (rM)*	CALL	Call subroutine
return (rM)	RETURN	Return from subroutine
do K, {L, rM}	DO	Do next K+1 instruction(s) L+1 (or rM+1) time(s). K=0, 1, 2, ...127; L=rM=0, 1, 2, ...2047
dolock K, {L, rM}	DO	*dolock* signals interlocked bus accesses.
doblock {L, rM}	DO	*doblock* signals quad-word transfers.
ireturn	IRETURN	Return from interrupt
sftrst	SFTRST	Soft Reset; Changes error level to base level; encoded as spc=(byte)r0
waiti	WAITI	Wait for interrupt; encoded as spc=(long)r0

* When using pc as rB, the conditional or unconditional goto and call instructions may be written with the syntax *pcgoto label* or *pccall label*, respectively. The linker will compute the appropriate N (offset) value to encode.

where:

rB = pc, r0-r22
rM = r1-r22
N = 16-bit signed integer
M= 24-bit unsigned integer
COND = one of the DSP3210 Condition Codes (See Tables below)

Table 4-7 Replacement for COND

(COND) - CAU Conditions		
Condition	**Flag***	**Meaning**
false	Not applicable	Condition is always false.
true	Not applicable	Condition is always true.
pl	n=0	Result is nonnegative (plus)
mi	n=1	Result is negative (minus)
ne	z=0	Result not equal to zero
eq	z=1	Result equal to zero
vc	v=0	Overflow clear, no overflow
vs	v=1	Overflow set, overflowed
cc	c=0	Carry clear, no carry
cs	c=1	Carry set, carry
ge	n^v=0	Greater than or equal to zero
lt	n^v=1	Less than zero
gt	z | (n^v)=0	Greater than zero
le	z | (n^v)=1	Less than or equal to zero
hi	c | z=0	Greater than (unsigned number)
ls	c | z=1	Less than (unsigned number)
(COND) - DAU Conditions		
Condition	**Flag**	**Meaning**
ane	Z=0	Not equal to zero
aeq	Z=1	Equal to zero
age	N=0	Greater than or equal to zero
alt	N=1	Less than zero
avc	V=0	Overflow clear, no overflow
avs	V=1	Overflow set, overflowed
auc	U=0	Underflow clear, no underflow
aus	U=1	Underflow set, underflowed
agt	N | Z=0	Greater than zero
ale	N | Z=1	Less than or equal to zero
(COND) - IO Conditions		
Condition	**Flag**	**Meaning**
ibe	ibf=0	Input buffer empty
ibf	ibf=1	Input buffer full
obe	obe=1	Output buffer empty
obf	obe=0	Output buffer full
syc	sy=0	Sync signal low
sys	sy=1	Sync signal high
fbc	fb=0	Serial frame boundary clear
fbs	fb=1	Serial frame boundary set
ir0s	ir0n=1	IR0N pin is logic one (1).
ir0c	ir0n=0	IR0N pin is logic zero (0).
ir1s	ir1n=1	IR1N pin is logic one (1).
ir1c	ir1n=0	IR1N pin is logic zero (0).

* Symbol interpretation: ^ = XOR; |= OR

4.3.2 CA Instructions - Arithmetic/Logic

The CA arithmetic/logic instructions perform integer arithmetic and logical operations in the CAU. Instructions of this type do not have memory references. The operands for these instructions are contained in the CAU registers or supplied in the instruction as immediate data. Operands that are in memory must first be moved to the CAU register array using a data move instruction (see 4.3.3 CA Instructions - Data Move). The size of the operation can be specified to be long (32-bits), or short (16-bits). **Long is the default and does not have to be specified.**

Table 4-8 lists the CA Arithmetic/Logic instructions.

Symbols in Table 4-8 can be replaced with:

rD = r0—r22
rD1 = r0—r20, r22
rS, rS1, rS2 = r0—r22
rS3 = r0—r22, pc, pcsh
N = 16-bit signed integer
COND = one of the DSP3210 Condition Codes (see Table 4-7 Replacement for COND)

Table 4-8 CA Arithmetic/Logic Instructions

Instruction	CAU Flag	Instr. Reference	Description
rD = {(long), (short)} rS3+N	nzvc	ADD	Three operand add with 16-bit sign-extended immediate
rD = (long) rS <<| N	nz00	SHIFT-OR	Shift 16-bit immediate left 16 places and OR
rD = {(long), (short)} rD - N	nzvc	SUBTRACT	Right subtract with immediate
rD = {(long), (short)} N - rD	nzvc	SUBTRACT	Left subtract with immediate
rD = {(long), (short)} rD & N	nz00	ADD	Logical AND with immediate
rD = {(long), (short)} rD | N	nz00	OR	Logical OR with immediate
rD = {(long), (short)} rD ^ N	nz00	EXCLUSIVE-OR	Logical XOR with immediate
rD = {(long), (short)} rD # N	nz0c	ADD-CARRY REV	Carry reverse add with immediate
rD = {(long), (short)} rD << N	nz00	SHIFT LEFT	Left shift by N (0≤N≤31)
rD = {(long), (short)} rD >> N	nz00	SHIFT RIGHT	Logical right shift by N (0≤N≤31)
rD = {(long), (short)} rD $>> N	nz00	SHIFT RIGHT-ARITH	Arithmetic right shift by N (0≤N≤31)
{(long), (short)} rD - N	nzvc	COMPARE	Compare with immediate
{(long), (short)} rD & N	nz00	BIT TEST	Bit Test with immediate
[if(COND)] rD = {(long), (short)} rS1 + rS2	nzvc	ADD	Conditional 3-register add
[if(COND)] rD = {(long), (short)} rS1 - rS2	nzvc	SUBTRACT	Conditional 3-register subtract
[if(COND)] rD = {(long), (short)} rS1 & rS2	nz00	ADD	Conditional 3-register AND
[if(COND)] rD = {(long), (short)} rS1 &~ rS2	nz00	AND-COMPLEMENT	Conditional 3-register AND with complement
[if(COND)] rD = {(long), (short)} rS1 | rS2	nz00	OR	Conditional 3-register OR
[if(COND)] rD = {(long), (short)} rS1 ^ rS2	nz00	EXCLUSIVE OR	Conditional 3-register XOR
[if(COND)] rD = {(long), (short)} rS1 # rS2	nz0c	ADD-CARRY REV	Conditional 3-register carry reverse add
[if(COND)] rD = {(long), (short)} rS1 << rS2	nz00	SHIFT LEFT	Conditional left shift by rS2 (0≤rS2≤31)
[if(COND)] rD = {(long), (short)} rS1 >> rS2	nz00	SHIFT RIGHT	Conditional logical right shift by rS2 (0≤rS2≤31)
[if(COND)] rD = {(long), (short)} rS1 $>> rS2	nz00	SHIFT RIGHT-ARITH	Conditional arithmetic right shift by rS2 (0≤rS2≤31)
[if(COND)] {(long), (short)} rS1 - rS2	nzvc	COMPARE	Conditional compare
[if(COND)] {(long), (short)} rS1 & rS2	nz00	BIT TEST	Conditional bit test
[if(COND)] rD = {(long), (short)} rS/2	nz00	SHIFT RIGHT-ARITH	Conditional arithmetic right shift by 1
[if(COND)] rD = {(long), (short)} rS>>1	nz00	SHIFT RIGHT	Conditional logical right shift by 1
[if(COND)] rD = {(long), (short)} rS<<1	nz00	SHIFT LEFT	Conditional left shift by 1
[if(COND)] rD = {(long), (short)} rS*2	nzvc	ADD	Conditional multiply by 2
[if(COND)] rD = {(long), (short)} rS>>>1	nz0c	ROTATE RIGHT	Conditional rotate right through carry
[if(COND)] rD = {(long), (short)} rS<<<1	nzvc	ROTATE LEFT	Conditional rotate left through carry
[if(COND)] rD = {(long), (short)} -rS	nzvc	SUBTRACT	Conditional negate
[if(COND)] rD = {(long), (short)} rS	nzvc	ADD	Conditional assignment
[if(COND)] rD1 = {(long), (short)} rS {+, –}1	nzvc	INCR/DECR	Conditional increment/ decrement
[if(COND)] sp = (long) sp {++, --}	nzvc	INCR/DECR	Conditional increment/ decrement of stack pointer by 4 (Interrupts are disabled for one instruction cycle)

4.3.3 CA Instructions - Data Move

The CA data move instructions are divided into four types: (1) immediate data sets of CAU registers, (2) data transfers between memory and CAU registers, (3) data transfers between IO registers and CAU registers, and (4) data transfers between IO registers and memory. The data can be moved as 8-bit (byte), 16-bit (half-word), or 32-bit (word). Table 4-9 lists the CA Data Move instructions and Table 4-10 shows the operand replacements.

When loading CAU registers from memory or an IO register, the size of the transfer may be specified to be 8-bit (byte, hbyte, char), 16-bit (short, ushort), or 32-bit (long). **Long is the default and does not have to be specified.** The affect on CAU register bits for each operation is described below:

Destination CAU Register Bits Affected					
Type Specified	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	0	0	0	DATA	Zero-extend to upper 24-bits
(hbyte)	0	0	DATA	0	Zero-fill bits 7-0, zero-extend upper 16-bits
(char)	S	S	S	DATA	Sign-extend bit 7 to upper 24-bits
(short)	S	S	DATA	DATA	Sign-extend bit 15 to upper 16-bits
(ushort)	0	0	DATA	DATA	Zero-extend to upper 16-bits
(long)	DATA	DATA	DATA	DATA	Load bits 31—0.

When storing CAU registers in memory or IO registers, the size of the operation may be specified to be 8-bit (byte, hbyte), 16-bit (short), or 32-bit (long). **Long is the default and does not have to be specified.** When the destination is an IO register, only the number of IO register bits specified by the size of the transfer are affected (8, 16, 32). The size specified for the transfer must be the same as the size of the register implemented in the DSP3210. The difference between (byte) and (hbyte) is the CAU register bits selected for the operation. The CAU register bits selected for each operation are described below:

Source CAU Register Bits Selected					
Type Specified	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	X	X	X	DATA	Store bits 7—0
(hbyte)	X	X	DATA	X	Store bits 15—8
(short)	X	X	DATA	DATA	Store bits 15—0
(long)	DATA	DATA	DATA	DATA	Store bits 31—0

When loading IO registers from memory, the size of the transfer may be specified to be 8-bit (byte), 16-bit (short), or 32-bit (long). Only the number of IO register bits specified by the size of the transfer are affected (8, 16, 32). When storing IO register to memory, the size of the transfer may be specified to be 8-bit (byte), 16-bit (short), or 32-bit (long). The size specified for the transfer must be the same as the size of the register implemented in the DSP3210.

Table 4-9 CA Data Move Group Instructions

Instruction	CAU Flag	Instr. Reference	Description
rD= (short) N	nz00	SET	Reg Set- 16-bit sign extended to 32-bits
rD= (ushort24) M	—	SET24	Reg Set- 24-bit zero extended to 32-bits
{MEM, *L} ={(byte), (hbyte), (short), (long)} rS	—	STORE	Reg Store- MEM, *L are 8-, 16-, or 32-bit
rD = {(char), (byte), (hbyte)} {MEM, *L}	nz00	LOAD	Reg Load- MEM, *L are 8-bits
rD = {(short), (ushort)} {MEM, *L}	nz00	LOAD	Reg Load- MEM, *L are 16-bits
rD = (long) {MEM, *L}	nz00	LOAD	Reg Load- MEM, *L are 32-bits
iorD = {(byte), (hbyte), (short), [(long)]} rS	—	STORE	Reg Store- iorD are 8--, 16-, or 32-bit
rD = {(char), (byte), (hbyte)} iorS	nz00	LOAD	Reg Load- iorS are 8-bits
rD = {(short), (ushort)} iorS	nz00	LOAD	Reg Load- iorS are 16-bits
rD = (long) iorS	nz00	LOAD	Reg Load- iorS are 32-bits
MEM = {(byte), (short), (long)} iorS	—	STORE-IOR	IO Reg to Memory move
iorD = {(byte), (short), (long)} MEM	—	LOAD-IOR	Memory to IO Reg move

Table 4-10 Replacement for CA Data Move Instructions

	Value	Meaning
rS	r0-r22, pc, pcsh	One of 22 CAU registers, pc, or pc shadow register
rD	r0-r22	One of 22 CAU registers
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location rP is a memory pointer (rP=r0—r22); when using the sp (r21), the size of the transfer must be long (++ and -- are +4 and –4), rI is an increment register (rI=r0-r22)
iorS	ior#	One of the IO registers. The size of the selected register and transfer size must be the same. (#=0, 8, 10, 12, 14, 15)
iorD	ior#	One of the IO registers. (#=0, 8, 10, 12, 14, 15)
N	16-bit number	2's complement number
L	16-bit number	Unsigned number–used as a direct address
M	24-bit number	Unsigned number

4.4 DSP3210 Programming

This chapter explains how to combine the instructions presented in Chapter 3 to form valid DSP3210 programs. The programmer must be aware of the latencies that occur because of the pipelining of the device and the restrictions on the order of certain instructions. This chapter presents this information, along with a set of programming examples for encoding common functions.

4.4.1 Restrictions

This section describes three sequences of instructions that are not allowed. The assembler recognizes these sequences and generates a fatal error.

4.4.1.1 Restriction 1 - Z-Field Pointer

The instruction immediately following a DA instruction with a Z-field write, in which the Z-field pointer is post-modified and cannot reference the Z-field pointer except as another Z-field pointer of a DA instruction. This is a fatal error, and the assembler produces an error message. In the event that an instruction with a post-modified Z-field pointer follows a branch instruction, the assembler produces a warning message.

Example 1:

```
*r1++ = a2 = a2 + *r3 * a1
 r1 = r1 + 12
```

ERROR: Using reg from preceding inst's Z-field pointer

Example 2:

```
*r1-- = a2 = a2 + *r3 * a1
a2 = a2 + *r1 * a1
```

ERROR: Using reg from preceding inst's Z-field pointer

Example 3:

```
 if (le) goto r20
*r1 = a2 = a2 + a3 * a1
```

Warning: Use output pointer reg only as output pointer in DAU inst atdest after branch instruction.

4.4.1.2 Restriction 2 - CAU and IO Register Store

A CAU or IO register store instruction cannot immediately follow a DA instruction (multiply/accumulate or special function) that references memory in the Y field. This is a fatal error, and the assembler produces an error message. In the event that a DA instruction with a memory reference in the Y field immediately follows a branch instruction, the assembler produces a warning message. The affected CAU and IO Register Store instructions are:

```
{MEM, *L} ={(byte), (hbyte), (short), (long)} rS
iorD = {(byte), (hbyte), (short), [(long)]} rS
MEM = {(byte), (short), (long)}  iorS
```

Recall that one form of the DA instructions is:

```
[Z] = aN = [–]aM {+, –} Y ** X
```

Example 1:

```
a2 = a2 + *r2 *a3
*r4 = r5
```

ERROR: CAU reg or IO reg store can't follow DAU inst with 'Y' mem ref

Example 2:

a0 = ic(ibuf)
*r4 = r5

ERROR: CAU reg or IO reg store can't follow DA inst with 'Y' IO register ref

Example 3:

if (ge) goto r3
a2 = a2 + *r2 * a3

Warning: branch dest can't start with CAU reg store inst.

4.4.1.3 Restriction 3 - CAU and IO Register Load

A CAU register (r0—r22) that is being loaded from memory or from an IO register, or an IO register (ps, emr, pcw, dauc) that is being loaded from memory cannot be referenced in the following instruction. The only exception is that a CAU register can be referenced as the Z-field pointer of a DA instruction. Note that the flags that result from this load cannot be referenced in the following instruction either. This is a fatal error, and the assembler produces an error message. If a CAU or IO register load follows a branch instruction, the assembler produces a warning message. A dauc load from memory must be followed by a CA instruction. The affected CAU and IO Register Load instructions are:

rD = {(char), (byte), (hbyte), (short), (ushort), (long)} {MEM, *L}
iorD = {(byte), (short), (long)} MEM

Example 1:

r3 = *r2
r4 = r3 + r7

ERROR:Cannot reference reg loaded in previous inst except as Z-field ptr

Example 2:

if (ne) goto r3 + 8
r4 = *r7++

Warning: branch destination can't start with inst referencing reg being loaded.

4.4.2 Latency Effects

The pipelined execution of Data Arithmetic DSP3210 instructions results in four latency effects that are listed in this section. See 8.2 Data Arithmetic Unit (DAU) for an explanation of the DAU pipeline.

4.4.2.1 Latency 1 - DA Instruction - Memory Writes

When a DA instruction specifies a write to memory, the value written is not available to be read from that location until four instructions later (three-instruction latency).

Example:

```
I1   *r3 = a0 = a0
I2   *r3 = a3 = a3
I3   .
I4   .
I5      a1 = *r3
```

The value read from memory in instruction five (I5) is the value written in I1, not I2.

4.4.2.2 Latency 2 - Accumulator as Multiplier Input

When an accumulator, a0-a3, is used as an input to the multiplier, its value is established no sooner than three instructions prior to the multiply instruction (2-instruction latency). Note that this also applies to an accumulator using the X field of an instruction of the form:

[Z =] aN = [–]Y {+, –} X

Example:

```
I1   a0 = a0 + *r1 * *r2
I2   a0 = a0 + a1
I3   .
I4   a2 = a0 * a0
I5   a1 = a2 + a0
```

The value of a0 used in I4 is calculated in I1. The value of a0 used in I5 is calculated in I2. Note that the value of a2 used in I5 is calculated in I4 since there is no latency effect on accumulators used as inputs to the adder.

4.4.2.3 Latency 3 - Branching

When a CA Control Group instruction of the form *if()goto, call, return, goto* is executed, the instruction immediately following is also executed before the branch occurs. This is commonly referred to as a delayed branch. The *ireturn* instruction is different, and execution of the base-level program resumes in the following instruction cycle.

Example 1:

```
I1   if (eq) goto over
I2   r1 = 3
I3   .
```

The instruction I2 is executed even if the condition is true and the branch is taken. If this is undesirable, a *nop* can be placed after the branch instruction, or if possible, the instructions can be rearranged. Because of this latency, a complex situation arises if successive branch instructions are coded as in the following example.

Example 2:

```
I1   goto A
I2   goto B
     .
     .
A:   .
     .
     .
B:   .
     .
C:   .
     .
```

The order of execution is I1, I2, A, B. If the instruction at A is not a goto, execution continues from B. If the instruction at A is goto C, the order of execution is I1, I2, A, B, C, and execution continues from C. Successive branch instructions are useful in some applications.

4.4.2.4 Latency 4 - Conditional Branching on DAU Conditions

A DAU condition tested by a conditional branch or conditional arithmetic/logic instruction is established by the last DA instruction that affects DAU flags no sooner than four instructions prior to the test (3-instruction latency).

Example:

```
I1   a0 = a0 + a1
I2   a2 = a0 * a2
I3   .
I4   .
I5   if (agt) goto next
I6   .                /* latent instruction */
```

The condition tested in I5 is established by the instruction at I1, not I2.

Because of this latency effect, use the zero-latency *ifalt, ifagt, and ifaeq* functions where possible (see 4.2.2 DA Instructions - Special Functions). The DA condition tested by these conditional accumulator loads is established by the last DA instruction that affected the DAU flags.

4.4.3 Common Programming Techniques

The following are programming techniques common to the DSP3210.

4.4.3.1 Subroutine Nesting

The following example is intended only as an illustration of DSP3210 programming technique. The convention adopted is that the stack frame grows from high addresses toward low addresses, and that the stack pointer contains the first empty location. The latter uncommon convention saves one instruction of overhead in the push operation because the DSP3210 has only postincrement and not preincrement addressing.

The following example shows how to nest subroutines:

```
        call subr1 (r1)          /* ret1 is stored in r1 */
        .                        /* latent instruction   */
ret1:   .
        .
        .
subr1:  .
        .
        .
         *sp-- = r1              /* push ret1 onto stack using stack pointer*/
         call subr2 (r1)         /* ret2 is stored in r1 */
        .                        /* latent */
ret2:   .
        .
        .
        return (r1)              /* go to ret1 */
        .                        /* latent    */
subr2:  .
        .
        .
        sp = sp++                /* sp now points to top of stack    */
         r1 = *sp                /* restore linkage pointer (ret1) */
         return (r1)             /* go to ret2  */
        .                        /* latent */
```

Alternatively, if only a small level of nesting is required and sufficient CAU registers are available, the different return addresses can simply be saved in different CAU registers.

4.4.3.2 Passing Parameters

Parameters may be passed to a subroutine if they immediately follow the latent instruction associated with the call:

```
1.  call subr (r1);
2.  .                    /* latent instruction */
3.  parameter #1 for subr
4.  parameter #2 for subr
5.  parameter #3 for subr
6.  parameter #4 for subr
7.  .
etc.
```

The subroutine should use *r1++ to retrieve the parameters. After all four parameters are retrieved, a *return (r1)* is executed that resumes execution at line 7.

4.4.3.3 Quick Interrupt Facility

Each entry in the interrupt vector table may contain two instructions. In general, these may be used for a *goto* and the delayed branch latent instruction. Alternatively, by using these two locations for an instruction followed by an *ireturn*, an interrupt routine with an overhead of three instruction cycles can be accomplished.

The interrupt operation performed by the DSP3210 while processing the instruction stream I1, I2, and I3 is as follows:

```
I1                  /* program instruction #1                          */
I2                  /* program instruction #2                          */
goto evtp + offset  /* overhead branching to interrupt                 */
nop                 /* vector table plus the offset is computed by the DSP3210 based on  */
                    /* the interrupt source                            */
IR1                 /* useful interrupt routine instruction */
ireturn             /* no latent instruction executed after ireturn                      */
I3                  /* program instruction #3                          */
```

IR1 is the useful interrupt routine instruction. **IR1 cannot be a DA instruction.**

4.5 Common Signal Processing Functions

This section presents instruction sequences for common functions performed in digital signal processing. Many more example programs are contained in the DSP3210 Application Software Library Reference Manual. **All instructions require one 32-bit word for storage and execute in one instruction cycle.** Note that the time to execute one instruction is affected by conflict and external waitstates.

4.5.1 Standard Signal Processing Functions

The following functions illustrate the source code syntax and programming techniques for the DSP3210. All instruction sequences may be preceded by the *do* instruction for repeated execution.

(a) Convolution (FIR filter)

$$y(n) = \sum_{k=0}^{k=N-1} h[k] * x(n-k)$$

```
(1)     a1 = a1 + (*r3++ = *r4++) * *r2++
```

(b) LMS update for adaptive filtering

$$h(k+1) = h(k) + 2 * \mu * e(k) * x(k)$$

```
(1)     *r2++ = a1 = *r2 + a0 * *r4++
```

(c) Second-order biquad IIR filter

$$w_i(n) = v_{i-1}(n) - d(i, 1)s_{i1}(n-1) - d(1, 2)s_{i2}(n-1)$$
$$v_i(n) = n(i, 0)w_i(n) + n(i, 1)s_{i1}(n-1) + n(i, 2)s_{i2}(n-1)$$
$$s_{i1}(n) = w_i(n)$$
$$s_{i2}(n) = s_{i1}(n-1)$$

```
(1)     a0 = * r5 * *r2++                          /* input =* K1      */
(2)     a0 = a0 - *r3++ * *r2++                    /* d(i, 1) term     */
(3)     *r4++ = a0 = a0 - *r3-- * *r2++            /* d(i, 2) term     */
(4)     a0 = a0 + (*r4++ = *r3++) * *r2++          /* n(i, 1) term     */
(5)     *r6 = a0 = a0 + *r3++ * *r2++              /* n(i, 2) term     */
```

(d) FFT Butterfly

```
(1)   a0 = *r1++r17 + a2 * *r3++r17          /* T = R(i) * Ur * R(k)      */
(2)   *r11++r17 = a0 = a0 - *r3++r19 * a3    /* R(i) = T = T - Ui * I(k)  */
(3)   a1 = *r1++r19 + a3 * *r3++r17          /* T = I(i) + Ui * R(k)      */
(4)   *r12++r17 = a0 = -a0 + *r1++r17 * *r6  /* (Rk) = -T + 2 * R(i)      */
(5)   *r11++r15 = a1 = a1 + *r3++r17 * *a2   /* I(i) = T = T + Ur * I(k)  */
(6)   *r12++r15 = a1 = -a1 + *r1++r15 * *r6  /* I(k) + -T + 2 * I(i)      */
```

(e) In-place bit reversal

```
A:    r1 - r2                        /* compare true and bit-reversed  */
      if(ge) pcgoto B                /* exchange if r1 < r2            */
      r5 = r1                        /* latent instruction             */
      *r1++r17 = a0 = *r2++r17       /* complex exchange               */
      *r1++r19 = a0 = *r2++r19
      *r2++r19 = a0 = *r5++r19
      *r2++r19 = a0 = *r5++r19
B:    r1 = r1 + 4
      if(r18-- >= 0) pcgoto A
      r2 = r2 # r16                  /* # is carry reverse add         */
```

(f) Inverse using the seed instruction

```
(1)   a3 = seed(*r1)                 /* r1 points to input table       */
(2)   r3 = _K                        /* r3 points to constants         */
(3)   r4 = result                    /* r4 points to output table      */
(4)   a0 = *r3++ + a3 * *r1          /* a0 = -1.6 - x * y              */
(5)   a1 = *r3++ * a3                /* a1 = 0.8 * x                   */
(6)   nop
(7)   a0 = *r3++ + a0 * a0           /* a0 = 31 - 3.2 * x * y + x^2 * y^2 */
(8)   a2 = a1 * *r1++                /* a2 = 0.8 * x * y               */
(9)   nop
(10)  a0 = a0 * a1                   /* a0 = 2.8 - 2.6 * x^2 * y + 0.8 *  */
                                     /* x^3 * y^2                         */
(11)  a1 = *r3--r17 + a0 * a2        /* a1 = 2.8 * x * y - 2.6 * x^2 *    */
                                     /* y^2 + 0.8 * x^3 * y^3             */
(12)  nop
(13)  nop
(14)  *r4++ = a0 = a0 - a1 * a0      /* result is in a0                   */

_K: float -1.60138661
    float 0.82371354
    float 0.85221935
    float -1.0
```

4.5.2 Nonlinear Functions

Several nonlinear functions are programmed using the *ifalt or ifagt* special functions (see 4.2.2 DA Instructions Special Functions). The DSP3210 programs for some of these nonlinear functions are included here.

(a) Absolute Value or Full-Wave Rectification

```
(1)   a0 = -a2
(2)   a2 = ifagt(a0)
```

The value in a2 is replaced by its absolute value.

(b) Signum

```
(1)    a0 = *r1
(2)    a1 = *r3
(3)    *r3 = a0 = ifalt(*r2)
```

The signum, or sign function, returns a +1 for positive operands and a –1 for negative operands. The above program assumes that the memory location pointed to by r1 contains a floating-point +1 and that the location pointed to by r2 contains a floating-point –1. The value in the memory location pointed to by r3 is replaced by its signum. (Note: duac[6]=0, therefore Z-field writes are unconditional).

(c) Minimum

```
(1)    a0 = –a1 + a2
(2)    a1 = ifalt(a2)
```

Accumulator a1 is written with the smaller of the two values stored in a1 and a2.

(d) Maximum

```
(1)    a0 = a1 – a2
(2)    a1 = ifalt(a2)
```

Accumulator a1 is written with the larger of the two values in a1 and a2.

(e) Half-Wave Rectification

```
(1)    a1 = *r2
(2)    a1 = ifalt(*r1)    /* r1 points to a floating-point zero */
```

The contents of memory location r2 are half-wave rectified and then stored in a1.

(f) Two-Sided Limiter

```
(1)    a2 = –*r1          /* initialize a2 with negative limit      */
(2)    a1 = –a0 + *r1     /* compare a0 with positive limit         */
(3)    a0 = ifalt(*r1)    /* if a0>limit, replace a0 with limit     */
(4)    a1 = a0 + *r1      /* compare a0 with negative limit         */
(5)    a0 = ifalt(a2)     /* if a0<limit, replace a0 with neg. limit */
```

The two-sided limiter restricts the magnitude of the input. In the example above, it is assumed that the limit is stored in a memory location pointed to by r1. The contents of a0 are limited, and the result is stored back into a0.

4.6 Detailed Description of Instruction Set

This section contains a detailed description of the DSP3210 instruction set presented in sections 4.2 and 4.3. The description consists of a short explanation of the operation performed by the instruction, the flags affected by the instruction, a description of the binary encoding, and an example. The instructions are organized alphabetically by the instruction reference name.

Instruction Reference	Page	Instruction Reference	Page
ADD	4-21	IFALT	4-54
ADD-CARRY REV	4-24	INCR/DECR	4-55
AND	4-25	INT16	4-56
AND-COMPLEMENT	4-26	INT32	4-57
BIT TEST	4-27	IRETURN	4-58
CALL	4-28	LOAD	4-59
COMPARE	4-30	LOAD-IOR	4-61
DO	4-31	NOP	4-62
DSP	4-33	OC	4-63
EXCLUSIVE OR	4-34	OR	4-64
FADD-STORE	4-35	RETURN	4-65
FADD-TAP	4-36	ROTATE LEFT	4-66
FLOAD-STORE	4-37	ROTATE RIGHT	4-67
FLOAT16	4-38	ROUND	4-68
FLOAT32	4-39	SEED	4-69
FMULT-ACC-STORE	4-40	SET	4-70
FMULT-ACC-TAP	4-41	SET24	4-71
FMULT-ADD-STORE	4-42	SFTRST	4-72
FMULT-STORE	4-43	SHIFT LEFT	4-73
FMULT-TAP	4-44	SHIFT-OR	4-75
GOTO	4-45	SHIFT RIGHT	4-76
GOTO-COND	4-47	SHIFT RIGHT-ARITH	4-78
GOTO-LOOP	4-49	STORE	4-81
IC	4-50	STORE-IOR	4-82
IEEE	4-51	SUBTRACT	4-83
IFAEQ	4-52	WAITI	4-85
IFAGT	4-53		

INSTRUCTION:

rD = {[(long)], (short)} rS3 + N (1)
[if(COND)] rD = {[(long)], (short)} rS1 + rS2 (2)
[if(COND)] rD = {[(long)], (short)} rS * 2 (3)
[if(COND)] rD = {[(long)], (short)} rS (4)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

These instructions perform integer add operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. A (short) operation sign extends the result from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected.

(1) is a three-operand add instruction with an immediate operand. The contents of rS are added to a value N, and the result is placed in rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits. This instruction is useful in computing addresses by the base plus displacement technique.

(2) is a 3-register add instruction. The contents of register rS1 and register rS2 are added, and the result is placed in register rD.

(3) is a multiply by 2 instruction. This instruction multiplies the contents of register rS by two, and the result is placed in register rD. This instruction is performed by adding rS to itself (rS+rS). This instruction should be used for a left shift by 1 when setting of the carry flag is desired.

(4) is an assignment instruction. This instruction assigns the contents of register rS to the register rD. The operation actually performed is rD = rS+r0.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer
rS3	r0—r22, pc, pcsh	r0 (hard-wired zero), or one of 22 CAU registers, the program counter, or the program counter shadow register

INSTRUCTION (cont.): rD = {[(long)], (short)} rS3 + N
[if(COND)] rD = {[(long)], (short)} rS1 + rS2
[if(COND)] rD = {[(long)], (short)} rS * 2
[if(COND)] rD = {[(long)], (short)} rS

Replacement Table for CA Instructions (COND) - CAU Conditions		
Condition	**Flag†**	**Meaning**
false	Not applicable	Condition is always false.
true	Not applicable	Condition is always true.
pl	n=0	Result is nonnegative (plus)
mi	n=1	Result is negative (minus)
ne	z=0	Result not equal to zero
eq	z=1	Result equal to zero
vc	v=0	Overflow clear, no overflow
vs	v=1	Overflow set, overflowed
cc	c=0	Carry clear, no carry
cs	c=1	Carry set, carry
ge	n^v=0	Greater than or equal to zero
lt	n^v=1	Less than zero
gt	z | (n^v)=0	Greater than zero
le	z | (n^v)=1	Less than or equal to zero
hi	c | z=0	Greater than (unsigned number)
ls	c | z=1	Less than (unsigned number)

† Symbol interpretation: ^ = XOR; |= OR

Replacement Table for CA Instructions (COND) - DAU Conditions		
Condition	**Flag**	**Meaning**
ane	Z=0	Not equal to zero
aeq	Z=1	Equal to zero
age	N=0	Greater than or equal to zero
alt	N=1	Less than zero
avc	V=0	Overflow clear, no overflow
avs	V=1	Overflow set, overflowed
auc	U=0	Underflow clear, no underflow
aus	U=1	Underflow set, underflowed
agt	N | Z=0	Greater than zero
ale	N | Z=1	Less than or equal to zero

INSTRUCTION (cont.): rD = {[(long)], (short)} rS3 + N
[if(COND)] rD = {[(long)], (short)} rS1 + rS2
[if(COND)] rD = {[(long)], (short)} rS * 2
[if(COND)] rD = {[(long)], (short)} rS

Replacement Table for CA Instructions (COND) - IO Conditions		
Condition	**Flag**	**Meaning**
ibe	ibf=0	Input buffer empty
ibf	ibf=1	Input buffer full
obe	obe=1	Output buffer empty
obf	obe=0	Output buffer full
syc	sy=0	Sync signal low
sys	sy=1	Sync signal high
fbc	fb=0	Serial frame boundary clear
fbs	fb=1	Serial frame boundary set
ir0s	ir0n=1	IR0N pin is logic one (1).
ir0c	ir0n=0	IR0N pin is logic zero (0).
ir1s	ir1n=1	IR1N pin is logic one (1).
ir1c	ir1n=0	IR1N pin is logic zero (0).

LATENCY

A DAU condition tested by a conditional branch or conditional arithmetic/logic instruction is established by the last DA instruction that affects DAU flags no sooner than four instructions prior to the test. For details on latency effects refer to Secion 4.4.2.

INSTRUCTION FORMAT (1) (Format 5a—b)

Bit	31	30—26	25—21	20—16	15—0
Field	E	0 0 1 0 1	rD	rS3	N

INSTRUCTION FORMAT (2–4) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2, rS3 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (+).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r11= (short) r1 + 32

Bit	31	30—26	25—21	20—16	15—0
Encoding	0	0 0 1 0 1	0 1 0 1 1	0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
Meaning	(short)		r11	r1	0x0020

ADD-CARRY REV

INSTRUCTION: rD = {[(long)], (short)} rD # N (1)
[if(COND)] rD = {[(long)], (short)} rS1 # rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzc, v=0

DESCRIPTION

These instructions perform carry reverse ADD operations, i.e., an ADD operation with the carry propagating from MSB to LSB. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. The (short) operation propagates the carry from bit 15 to bit 0 and sign extends the result from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a carry reverse add operation on the contents of register rD with the value N and the result is placed in the register rD. N is a 16-bit 2's complement number MSB-extended to 32-bits.

(2) This instruction performs a carry reverse add operation on the contents of register rS1 with the contents of register rS2, and the result is placed in the register rD. If the instruction is conditionally executed and the COND is true, the result of the operation on rS1 and rS2 is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (See page 4-21 ADD). When rS1 = rS2 (rS), this instruction performs a right shift by one binary place and the LSB of rS is placed in the **c** flag.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (#).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r2 = r14 # r4

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	0 0 1 1	0 0 0 1 0	0 1 1 1 0	0 0 0 0 0 1	0 0 1 0 0
Meaning	(long)		#	r2	r14	true	r4

INSTRUCTION: rD = {[(long)], (short)} rD & N (1)
[if(COND)] rD = {[(long)], (short)} rS1 & rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform bit-by-bit, logical AND operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a bit-by-bit, logical AND operation on the contents of register rD with the value N and the result is placed in the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) This instruction performs a bit-by-bit, logical AND operation on the contents of register rS1 with the contents of register rS2, and the result is placed in the register rD. If the instruction is conditionally executed and the COND is true, the result of the operation on rS1 and rS2 is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (See page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (&).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(cc) r1 = (short) r1 & r2

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	0	0 0 1 1 0 0	1 1 1 0	0 0 0 0 1	0 0 0 0 1	0 0 1 0 0 0	0 0 0 1 0
Meaning	(short)		&	r1	r1	cc	r2

AND-COMPLEMENT

INSTRUCTION: [if (COND)] rD = {[(long)], (short)} rS1 &~ rS2

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

This instruction performs a bit-by-bit, logical AND operation on the contents of register rS1 with the complement of the contents of register rS2. The result is placed in the register rD. The items in the brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) operation, the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (See page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
F Field: Specifies the ALU function to be performed (&~).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r5 = (short) r4 &~ r3

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	0	0 0 1 1 0 0	0 1 1 0	0 0 1 0 1	0 0 1 0 0	0 0 0 0 0 1	0 0 0 1 1
Meaning	(short)		&~	r5	r4	(true)	r3

INSTRUCTION: {[(long)], (short)} rD & N (1)
[if(COND)}] {[(long)], (short)} rS1 & rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform bit-by-bit, logical AND operation, but the results are not stored. These instructions can be used for bit tests. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a bit-by-bit, logical AND operation on the contents of register rD with the value N. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) This instruction performs a bit-by-bit, logical AND operation on the contents of register rS1 with the contents of register rS2. If the instruction is conditionally executed and the COND is true, the flags are set. If the instruction is conditionally executed and the COND is false, the flags are not affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (& (bit test)).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r5 & r13

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 1 1 1	0 0 0 0 0	0 0 1 0 1	0 0 0 0 0 1	0 1 1 0 1
Meaning	(long)		& (bit test)		r5	true	r13

CALL

INSTRUCTION: call {N, M, rB, rB+N} (rM)

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction transfers control of the program counter (pc) to a location specified in the braces {}. When this instruction is executed, the value of the program counter (pc) is stored in the specified rM register, and the value in the braces {} is moved to the pc. Register rM contains the address of the instruction that follows the latent instruction. The values in the braces {} specify an address N (16-bit MSB extended to 32-bits), an address M (24-bit zero extended to 32-bits), a CAU register rB, or a value in a CAU register rB plus a displacement value N (16-bit MSB extended to 32-bits). N is always 16 bits and is sign-extended to 32 bits for all operations. M is 24-bits and is zero-extended to 32-bits. When using pc as rB, the syxtax:

pccall label

may be used to perform a pc-relative subroutine call. The linker will compute the appropriate N (offset) value. The call M (rM) instruction is a format 8c instruction. All others are format 4a. **This instruction disables interrupts for one instruction cycle.**

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rB	pc, r0—r22	The program counter, r0 (hard-wired zero), or one of 22 CAU registers
rM	r1—r22	One of 22 CAU registers
N	16-bit number	2's complement integer
M	24-bit number	Unsigned integer

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 4a)

Bit	31—26	25—21	20—16	15—0
Field	0 0 0 1 0 0	rM	rB	N

rM Field: Specifies the register used to store the return address.
rB Field: Specifies the base address register. When N specifies 0, rB is the destination address.
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address. When rB specifies r0, the N field is an absolute address.

INSTRUCTION FORMAT (Format 8c)

Bit	31-29	28-21	20—16	15—0
Field	1 1 1	NE	rM	N

rM Field: Specifies the register used to store the return address.
NE Field: Specifies the most significant 8-bits of the 24-bit integer. NE concatonated with N forms the 24-bit unsigned integer (NE || N).
N Field: Specifies the least significant 16-bits of the 24-bit unsigned integer.

INSTRUCTION (cont.): call {N, M, rB, rB+N} (rM)

EXAMPLE (For field encodings refer to Chapter 10.)

call pc+0x0111 (r20)

Bit	31—26	25—21	20—16	15—0
Encoding	0 0 0 1 0 0	1 1 0 0 0	0 1 1 1 1	0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1
Meaning		r20	pc	0x0111

COMPARE

INSTRUCTION: {[(long)],(short)} rD - N (1)
[if(COND)] {[(long)],(short)} rS1 - rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

These instructions perform subtract operations, but the results are not stored. These instructions are used in compare operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. A (short) operation sign extends the result from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction subtracts the value N from the contents of register rD. N is a 16-bit, 2's complement number that is MSB-extended to 32 bits.

(2) This instruction subtracts the contents of register rS2 from register rS1. If the instruction is conditionally executed and the COND is true, the flags are set. If the instruction is conditionally executed and the COND is false, the flags are not affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (- (compare)).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

(long) r1 - 15

Bit	31	30—25	24—21	20—16	15—0
Encoding	1	0 0 1 1 0 1	0 1 1 1	0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
Meaning	(long)		(compare)	r1	15

INSTRUCTION: do K,{L,rM} (1)
dolock K,{L,rM} (2)
doblock {L,rM} (3)

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction performs the next K + 1 instructions L + 1(or rM + 1) times. The argument K is an integer from 0 to 127. The second argument can either be a register rM, or a value from 0 to 2047. If a register rM is specified, only the lower 11 bits of the register are used to specify the loop count. When K = 0, the instruction to be repeated is cached on-chip for the duration of the do loop. This eliminates the repeated fetch of the same instruction.

(1) The do instruction initiates a do loop sequence. DMA cycles and interrupts are supported during do loops. For interrupts, the iteration and instruction count registers are shadowed.

(2) The dolock instruction is used to assert an external control signal, LOCKN, for the duration of the do loop. Interrupts and DMA cycles are disabled during a dolock sequence.

(3) The doblock instruction is used to decompose a block move into quad-word transfers. At the beginning of each quad-word, the BLMN signal is asserted and the MS0—MS3 pins indicate a quad-word transfer size (Note that BLMN and MS0—MS3 are configurable, see Section 6.1 - Bus Interface Signal Descriptions). The only instruction allowed following the *doblock* instruction is *a0=(Z=Y)*a0*, therefore K = 0 is implicit. At the completion of the do block sequence, the DAU flags are undefined. Interrupts and DMAs are disabled during each quad-word transfer. See 6.6.3 Quad-word Block Move Indicators.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
K	0—127	Immediate value from 0 to 127
L	0—2047	Immediate value from 0 to 2047
rM	r1—r22	One of 22 CAU registers

LATENCY

The do, dolock, and doblock instructions have no latency properties. Instructions encompassed by a do loop execute with the same latencies and restrictions therefore interactions between the last instruction of the loop and the first instruction of the loop should be carefully inspected. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 3b)

Bit	31—25	24	23	22—18	17—11	10—0
Field	1 0 0 0 1 1 0	B	M		K	L

INSTRUCTION FORMAT (Format 3c)

Bit	31—25	24	23	22—18	17—11	10—5	4—0
Field	1 0 0 0 1 1 1	B	M		K		rM

B,M Fields: B specifies the dolock instruction, and M specifies the doblock instruction.
K Field: Specifies the number of instructions to be repeated (K field is encoded with one less than the desired number).
L Field: Specifies the iteration count (L field is encoded with one less than the desired count).
rM Field: Specifies the register containing the iteration count (the register value is one less than the desired iteration count).

INSTRUCTION (cont.): do K,{L,rM}
dolock K,{L,rM}
doblock {L,rM}

EXAMPLE (For field encodings refer to Chapter 10.)

do 4, r20

Bit	31—25	24	23	22—18	17—11	10—5	4—0
Encoding	1 0 0 0 1 1 1	0	0	0 0 0 0 0	0 0 0 0 1 0 0	0 0 0 0 0 0	1 1 0 0 0
Meaning					4		r20

INSTRUCTION: [Z =] aN = dsp(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction converts the Y operand, an IEEE format 32-bit floating-point number, to a DSP3210 format floating-point number, and the result is placed in an accumulator (aN). If the exponent of the IEEE number is 255, it is treated as positive or negative infinity according to its sign. If positive, the converted DSP3210 number is the largest positive representable number in the DSP3210 format, namely 0x7fffffff. If negative, the converted DSP3210 number is the most negative number representable in the DSP3210 format, namely 0x800000ff. If the exponent of the IEEE number is zero, the converted DSP3210 number is 0x00000000, or positive zero. The DSP3210 format does not recognize a negative zero. The denormalized IEEE numbers are not supported. If the IEEE format input operand is NaN (not a number), the NaN error condition is set. If a Z field is specified, the result is also stored to a word address in memory. **Note that the Y operand can not come from an accumulator.**

IEEE-to-DSP Conversion Special Cases			
Y†	aN	Flags Z N U V, NaN	Comments
s=0; exp=255; mant=0	0x7fffffff	0 0 0 1 0	Treated as +∞
s=1; exp=255; mant=0	0x800000ff	0 1 0 0 0	Treated as -∞
s=0; exp=255; mant ≠0	0x7fffffff	0 0 0 1 1	Non-zero mantissa treated as +∞
s=1; exp=255; mant ≠0	0x800000ff	0 1 0 1 1	Non-zero mantissa treated as -∞
s=0; exp= 0; mant ≠0	0x00000000	1 0 1 0 0	Denormalized number treated as zero
s=1; exp= 0; mant ≠0	0x00000000	1 0 1 0 0	Denormalized number treated as zero

† s = sign bit; exp = exponent; mant = mantissa

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5 - Refer to [Z=] aN = ieee(Y))

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++ = a1 = dsp(*r9++)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	1 1 0 1	0 1	0 0 0 0 0 0 0	1 0 0 1 1 1 1	0 0 0 1 1 1 1
Meaning		dsp	a1		*r9++	*r1++

EXCLUSIVE OR

INSTRUCTION: rD = {[(long)],(short)} rD ^ N (1)
[if(COND)] rD = {[(long),(short)]} rS1 ^ rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz,v=c=0

DESCRIPTION

These instructions perform bit-by-bit, logical XOR (exclusive OR) operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a bit-by-bit, logical XOR operation on the contents of register rD with the value N and the result is placed in the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) This instruction performs a bit-by-bit, logical XOR operation on the contents of register rS1 with the contents of register rS2, and the result is placed in the register rD. If the instruction is conditionally executed and the COND is true, the result of the operation on rS1 and rS2 is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (^).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(ir0c) r2 = r3 ^ r4

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 0 0 0	0 0 0 1 0	0 0 0 1 1	1 0 1 1 0 0	0 0 1 0 0
Meaning	(long)		^	r2	r3	ir0c	r4

INSTRUCTION: [Z =] aN = [–] Y {+,–} X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

In this instruction, the operand specified by the X field is added to or subtracted from the operand or minus the operand specified by the Y field and the result is stored in an accumulator (aN). If a Z field is specified, the result is also be stored to a word address in memory. The operation actually performed is [Z =] aN = [–] Y {+,–} 1.0 * X.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 1)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 0 1	M	0	F	S	N	X	Y	Z

M Field: Specifies a constant value to be used for multiplier input (1.0).
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X Field: Specifies the multiplier input via accumulator direct or register indirect addressing modes.
Y Field: Specifies the adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a1 = *r5-- - *r7++

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 0 1	1 0 1	0	0	1	0 1	0 1 1 1 1 1 1	0 1 0 1 1 1 0	0 0 0 0 1 1 1
Meaning		(1.0)		+	–	a1	*r7++	*r5--	No write

FADD-TAP

INSTRUCTION: aN = [–] (Z=Y) {+,–} X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

In this instruction, the operand specified by the X field is added to or subtracted from the operand specified by the Y field. The result or minus the result is stored in an accumulator (aN). The value of the Y operand is stored to a word address in memory. The operation actually performed is: aN = [–] (Z=Y) {+,–} 1.0 * X.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 4)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 0 1	1 1 0	0	F	S	N	X	Y	Z

F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X Field: Specifies the multiplier input via accumulator direct or register indirect addressing modes.
Y Field: Specifies the adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a1 = (*r13 = a3) - *r1--

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 0 1	1 1 0	0	0	1	0 1	0 0 0 1 1 1 0	0 0 0 0 0 1 1	1 1 0 1 0 0 0
Meaning		Format 4		+	-	a1	*r1--	a3	*r13

INSTRUCTION: [Z =] aN = [–] Y

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction places the operand or minus the operand specified by the Y field into an accumulator (aN). If a Z field is specified, the result is also be stored to a word address in memory. The operation actually performed is [Z=] aN = [–] Y + 0.0 * a0. If the Y operand is not a valid DSP3210 floating-point number, it will be treated as a diry-zero and the accumulator will be loaded with zero. See Section 3.3.3 32-bit Floating-Point Data Type. To perform memory-to-memory transfers use FMULT-TAP.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 1)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 0 1	M	0	F	S	N	X	Y	Z

M Field: Specifies a constant value to be used for multiplier input (0.0).
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product (+).
N Field: Specifies the accumulator used for result.
Y Field: Specifies the adder input via accumulator direct or register indirect addressing modes.
X Field: The default specification is a0 as the multiplier input.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1 = a2 = *r2--

Bit	31 30 29	28 27 26	25	24	23	22 21	20 -14	13 -7	6- 0
Encoding	0 0 1	1 0 0	0	0	0	1 0	0 0 0 0 0 0 0	0 0 1 0 1 1 0	0 0 0 1 0 0 0
Meaning		(0.0)		+	+	a2	(a0)	*r2--	*r1

FLOAT16

INSTRUCTION: [Z =] aN = float16(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZ,V=U=0

DESCRIPTION

This instruction converts the 16-bit 2's complement integer operand specified by the Y field to the equivalent 32-bit floating-point representation. The result is stored in an accumulator (aN). The integer operand Y can specify a 16-bit memory location or an accumulator. When the post-increment or post-decrement addressing mode is used for the Y operand the value of the modifier is +2 or -2, respectively. If a Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	16-bit memory location. ++ is +2 and -- is -2. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (float16).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++r15 = a3 = float16(*r2)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 0 1 0	1 1	0 0 0 0 0 0 0	0 0 1 0 0 0 0	0 0 0 1 0 0 1
Meaning		float16	a3		*r2	*r1++r15

INSTRUCTION: [Z =] aN = float32(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZ,V=U=0

DESCRIPTION

This instruction converts the 32-bit integer operand Y to the equivalent 32-bit floating-point representation. The result is stored in the accumulator aN and may optionally be written to memory. The 32-bit integer operand Y is located in a 32-bit memory location. If a Z field is specified, the result is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (float32).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r5++r17 = a3 = float32(*r2)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	1 0 1 0	1 1	0 0 0 0 0 0 0	0 0 1 0 0 0 0	0 1 0 1 0 1 1
Meaning		float32	a3		*r2	*r5++r17

FMULT-ACC-STORE

INSTRUCTION: [Z =] aN = [–] aM {+,- } Y*X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operands specified by the X and Y fields. This product is then added to or subtracted from the contents or minus the contents of an accumulator (aM), and the result is stored in an accumulator (aN). If the Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN, aM	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 3)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 1 1	M	0	F	S	N	X	Y	Z

M Field: Specifies the accumulator used for adder input.
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X,Y Fields: Specifies the multiplier inputs via accumulator direct or register indirect addressingmodes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++ = a0 = a1 – *r2-- * *r3++r17

Bit	31 30 29	28 27 26	25	24	23	22 21	20 —14	13 —7	6—0
Encoding	0 1 1	0 0 1	0	0	1	0 0	0 0 1 1 0 1 1	0 0 1 0 1 1 0	0 0 0 1 1 1 1
Meaning		a1		+	–	a0	*r3++r17	*r2--	*r1++

INSTRUCTION: aN = [–] aM {+,–} (Z=Y) * X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operands specified by the X and Y fields. The result of the multiplication (Y * X) is added to or subtracted from the contents of an accumulator (aM), and the result is stored in an accumulator (aN). The value of the Y field operand is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN, aM	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 2)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 1 0	M	0	F	S	N	X	Y	Z

M Field: Specifies the accumulator used for adder input.
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X,Y Fields: Specifies the multiplier inputs via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a0 = - a1 + (*r2-- = *r3++r17) * *r1++

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13 —7	6—0
Encoding	0 1 0	0 0 1	0	1	0	0 0	0 0 0 1 1 1 1	0 0 1 1 0 1 1	0 0 1 0 1 1 0
Meaning		a1		–	+	a0	*r1++	*r3++r17	*r2--

FMULT-ADD-STORE

INSTRUCTION: [Z =] aN = [–] Y {+,–} aM * X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operand specified by the X field with the contents of an accumulator (aM). This product is then added to or subtracted from the operand or minus the operand specified by the Y field and the result is stored in an accumulator (aN). If the Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN, aM	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 1)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 0 1	M	0	F	S	N	X	Y	Z

M Field: Specifies the accumulator used for multiplier input.
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X Field: Specifies the multiplier input via accumulator direct or register indirect addressing modes.
Y Field: Specifies the adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r2++r16 = a2 = *r14++ - a0 * *r10--

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 0 1	0 0 0	0	0	1	1 0	1 0 1 0 1 1 0	1 1 1 0 1 1 1	0 0 1 0 0 1 0
Meaning		a0		+	–	a2	*r10--	*r14++	*r2++r16

INSTRUCTION: [Z =] aN = [–] Y * X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operands specified by the X and the Y fields. The product or minus the product is stored in an accumulator (aN). If the Z field is specified, the result is also stored to a word address in memory. The operation actually performed is [Z=] aN = 0.0 {+,–} Y * X

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 3)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 1 1	M	0	F	S	N	X	Y	Z

M Field: Specifies a constant value to be used for adder input (0.0).
F Field: Specifies sign of operation for adder input (+).
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X,Y Fields: Specifies the multiplier inputs via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r2++r17 = a3 = *r12++ * *r11--

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 1 1	1 0 0	0	0	0	1 1	1 0 1 1 1 1 0	1 1 0 0 1 1 1	0 0 1 0 0 1 1
Meaning		0.0		+	+	a3	*r11--	*r12++	*r2++r17

FMULT-TAP

INSTRUCTION: aN = [–] (Z=Y) * X

CLASS: DA-Mulitply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operands specified by the X and Y fields. The product Y * X or minus the product is stored in an accumulator (aN). The value of the Y operand is stored to a word address in memory. The operation actually performed is: aN = 0.0 {+,–} (Z=Y) * X. This instruction can be used for 32-bit memory-to-memory transfers of arbitrary data, e.g. a0 = (Z=Y) * a0.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 2)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 1 0	M	0	F	S	N	X	Y	Z

M Field: Specifies a constant value to be used for adder input (0.0).
F Field: Specifies sign of operation for adder input (+).
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X,Y Fields: Specifies the multiplier inputs via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a0 = - (*r2++r16 = *r1) * *r8--

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 1 0	1 0 0	0	0	1	0 0	1 0 0 0 1 1 0	0 0 0 1 0 0 0	0 0 1 0 0 1 0
Meaning		(0.0)		+	–	a0	*r8--	*r1	*r2++r16

INSTRUCTION: goto {N, M, rB, rB+N, rB+M}

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction is an unconditional goto. When executed, the specified value in the braces {} is moved to the program counter (pc). The values in the braces {} specify either an address N (16-bit MSB extended to 32-bits), an address M (24-bit zero-extended to 32-bits), a CAU register rB, a value in a CAU register rB plus a displacement value N (16-bit MSB extended to 32-bits), or a value in a CAU register rB plus a displacement value M (24-bit zero extended to 32-bits). N is always 16 bits and is sign-extended to 32 bits for all operations. M is always 24-bits and it zero-extended to 32-bits. The goto M instruction is a format 8a instruction. The remaining instructions are encoded as format 0-1b, special cases of the *if (COND) goto* instruction. In this case, the branch is always taken. When using pc as rB, the syxtax:

pcgoto label

may be used to perform a pc-relative branch. The linker will compute the appropriate N (offset) value. **This instruction disables interrupts for one instruction cycle.**

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rB	pc, r0—r22	The program counter, r0 (hard-wired zero), or one of 22 general-purpose registers.
N	16-bit number	2's complement integer
M	24-bit number.	Unsigned integer

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. Normally, a nop is placed after the branch instruction. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 0—1b)

Bit	31—27	26—21	20—16	15—0
Field	1 0 0 0 0	C	rB	N

C Field: Specifies the condition (true).
rB Field: Specifies the base address register. When N specifies 0, rB is the destination address.
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address. When rB specifies r0, the N field is an absolute address.

INSTRUCTION FORMAT (Format 8a)

Bit	31-29	28-21	20—16	15—0
Field	1 0 1	NE	rB	N

rB Field: Specifies the base address register.
NE Field: Specifies the most significant 8-bits of the 24-bit integer. NE concatonated with N forms the 24-bit unsigned integer (NE || N).
N Field: Specifies the least significant 16-bits of the 24-bit unsigned integer.

GOTO

INSTRUCTION (cont.): goto {N, M, rB, rB+N}

EXAMPLE (For field encodings refer to Chapter 10.)

goto r1

Bit	31—27	26—21	20—16	15—0
Encoding	1 0 0 0 0	0 0 0 0 0 1	0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		true	r1	0x0000

INSTRUCTION: if(COND) goto {N, rB, rB+N}

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction is a conditional branch instruction that depends on the values of the flags. The condition (CAU COND, DAU COND, or IO COND) is selected from the tables below. CAU COND flags are established by the previous CA instruction that set flags. DAU COND flags are established by the most recent DA instruction that occurs four or more instructions prior to the condition test. IO COND flags can be affected by external events, and program or DMA activity. If the condition evaluates true, the value of the program counter (pc) is replaced with one of the values specified in the braces {}. These values include a CAU register rB, an address N (MSB extended), or a value in a CAU register rB plus a displacement value N (MSB extended). N is a 16-bit 2's complement number and is MSB-extended to 32 bits. When using pc as rB, the syxtax:

if (COND) pcgoto label

may be used to perform a pc-relative branch. The linker will compute the appropriate N (offset) value. **This instruction disables interrupts for one instruction cycle.**

Replacement Table for CA Instructions (COND) - CAU Conditions		
Condition	**Flag†**	**Meaning**
false	Not applicable	Condition is always false
true	Not applicable	Condition is always true
pl	n=0	Result is nonnegative (plus)
mi	n=1	Result is negative (minus)
ne	z=0	Result not equal to zero
eq	z=1	Result equal to zero
vc	v=0	Overflow clear, no overflow
vs	v=1	Overflow set, overflowed
cc	c=0	Carry clear, no carry
cs	c=1	Carry set, carry
ge	n^v=0	Greater than or equal to zero
lt	n^v=1	Less than zero
gt	z | (n^v)=0	Greater than zero
le	z | (n^v)=1	Less than or equal to zero
hi	c | z=0	Greater than (unsigned number)
ls	c | z=1	Less than (unsigned number)

† Symbol interpretation: ^ = XOR; |= OR

Replacement Table for CA Instructions (COND) - DAU Conditions		
Condition	**Flag**	**Meaning**
ane	Z=0	Not equal to zero
aeq	Z=1	Equal to zero
age	N=0	Greater than or equal to zero
alt	N=1	Less than zero
avc	V=0	Overflow clear, no overflow
avs	V=1	Overflow set, overflowed
auc	U=0	Underflow clear, no underflow
aus	U=1	Underflow set, underflowed
agt	N | Z=0	Greater than zero
ale	N | Z=1	Less than or equal to zero

INSTRUCTION (cont.): if(COND) goto {N, rB, rB+N}

Replacement Table for CA Instructions (COND) - IO Conditions		
Condition	**Flag**	**Meaning**
ibe	ibf=0	Input buffer empty
ibf	ibf=1	Input buffer full
obe	obe=1	Output buffer empty
obf	obe=0	Output buffer full
syc	sy=0	Sync signal low
sys	sy=1	Sync signal high
fbc	fb=0	Serial frame boundary clear
fbs	fb=1	Serial frame boundary set
ir0s	ir0n=1	IR0N pin is logic one (1).
ir0c	ir0n=0	IR0N pin is logic zero (0).
ir1s	ir1n=1	IR1N pin is logic one (1).
ir1c	ir1n=0	IR1N pin is logic zero (0).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rB	pc, r0—r22	The program counter, r0 (hard-wired zero), or one of 22 general-purpose registers.
N	16-bit number	2's complement integer

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. A DAU condition tested by a conditional branch or conditional arithmetic/logic instruction is established by the last DA instruction that affects DAU flags no sooner than four instructions prior to the test. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 0-1b)

Bit	31—27	26—21	20—16	15—0
Field	10000	C	rB	N

C Field: Specifies the condition.
rB Field: Specifies the base address register. When N specifies 0, rB is the destination address.
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address. When rB specifies r0, the N field is an absolute address.

EXAMPLE (For field encodings refer to Chapter 10.)

if(pl) goto r1+0x0800

Bit	31—27	26—21	20—16	15—0
Encoding	10000	000010	00001	0000100000000000
Meaning		pl	r1	0x0800

INSTRUCTION: if(rM– >=0) goto {N, rB, rB+N}

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction is primarily used as a loop counter. The value of rM is treated as a 32-bit number and is decremented each time the instruction is executed. If the value of rM is greater than or equal to zero, the value of the pc changes to one of the specified locations in the braces {}. If rM is less than zero, the next instruction is executed. The values in the braces {} specify a CAU register rB, an address N (MSB extended) or a value in a CAU register rB plus a displacement value N (MSB extended). N is always 16 bits and is sign-extended to 32 bits for all operations. When using pc as rB, the syxtax:

if (rM– >=0) pcgoto label

may be used to perform a pc-relative branch. The linker will compute the appropriate N (offset) value. **This instruction disables interrupts for one instruction cycle.**

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rB	pc, r0—r22	The program counter, r0 (hard-wired zero), or one of 22 general-purpose registers.
rM	r1—r22	One of 22 CAU registers.
N	16-bit number	2's complement integer

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 3a)

Bit Field	31—26	25—21	20—16	15—0
	0 0 0 0 1 1	rM	rB	N

rM Field: Specifies the register used for the loop counter.
rB Field: Specifies the base address register. When N specifies 0, rB is the destination address.
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address. When rB specifies r0, the N field is an absolute address.

EXAMPLE (For field encodings refer to Chapter 10.)

if(r1– >=0) goto r2 - 0x00ab

Bit Encoding Meaning	31—26	25—21	20—16	15—0
	0 0 0 0 1 1	0 0 0 0 1	0 0 0 1 0	1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1
		r1	r2	0xff55

IC

INSTRUCTION: [Z =] aN = ic(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZ,V=U=0

DESCRIPTION

This instruction converts the operand specified by the Y field, an 8-bit μ-law, A-law, or unsigned word (depending on the value of the dauc register), to an equivalent 32-bit floating-point number. The result is stored in an accumulator (aN). The Y operand can specify a byte memory location or an accumulator. A detailed description of data formats is presented in 8.2.4 - Type Conversions. If Y specifies an accumulator, the 8-bit word must reside in the most significant byte of the accumulator. If a Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	8-bit memory location. ++ is +1 and -- is -1. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ic).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++ = a0 = ic(*r2++)

Bit	31—27	26—23	22 21	20 —14	13 —7	6—0
Encoding	0 1 1 1 1	0 0 0 0	0 0	0 0 0 0 0 0 0	0 0 1 0 1 1 1	0 0 0 1 1 1 1
Meaning		ic	a0		*r2++	*r1++

INSTRUCTION: [Z =] aN = ieee(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction converts the Y operand, a DSP32C format floating-point number, to an IEEE format, floating-point number. The Y operand can specify a memory location or an accumulator. If the exponent of the DSP32C number is zero, the converted IEEE number is 0x00000000 or positive zero. DSP32C to IEEE conversion never results in NaN (not a number). If a Z field is specified, the result is also stored to a word address in memory. In order to get the IEEE number correctly formatted in memory, the Z write must be performed in the instruction that specifies the ieee conversion.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ieee).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r4-- = a1 = ieee(*r9++)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	1 1 0 0	0 1	0 0 0 0 0 0 0	1 0 0 1 1 1 1	0 1 0 0 1 1 0
Meaning		ieee	a1		*r9++	*r4--

IFAEQ

INSTRUCTION: [Z =] aN = ifaeq(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction stores the Y operand in accumulator aN only if the result of the **most recent** DA instruction is equal to zero. If it is not zero, aN will retain its old value. Regardless, the Y address calculations are performed (register postmodification is performed if specified). The Y operand can specify a word memory location or an accumulator. Based on dauc[6], the Z-field operation can be conditional. If dauc[6]=0, the Z-field operation, if specified, is not conditional. Note that if the test fails, the previous value of aN is stored to memory. If dauc[6] = 1, the write to memory, if specified, is also conditional, but the address postmodifications are always performed. If written, the value of aN is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ifaeq).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a0 = ifaeq(*r7)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 1 1 0	0 0	0 0 0 0 0 0 0	0 1 1 1 0 0 0	0 0 0 0 1 1 1
Meaning		ifaeq	a0		*r7	No write

INSTRUCTION: [Z =] aN = ifagt(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction stores the Y operand in accumulator aN if and only if the result of the **most recent** DA instruction is greater than zero. If it is not greater than zero, aN retains its old value. Regardless, the Y address calculations are performed (register postmodification is performed if specified). The Y operand can specify a word memory location or an accumulator. Based on dauc[6], the Z-field operation can be conditional. If dauc[6] = 0, the Z-field operation, if specified, is not conditional. Note that if the test fails, the previous value of aN is stored to memory. If dauc[6] = 1, the write to memory, if specified, is also conditional, but the address postmodifications are always performed. If written, the value of aN is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ifagt).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r13 = a1 = ifagt(*r9++r18)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 1 1 1	0 1	0 0 0 0 0 0 0	1 0 0 1 1 0 0	1 1 0 1 0 0 0
Meaning		ifagt	a1		*r9++r18	*r13

IFALT

INSTRUCTION: [Z =] aN = ifalt(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction stores the Y operand in an accumulator (aN) only if the result of the **most recent** DA instruction is negative. If it is not negative, aN retains its old value. Regardless, the Y address calculations are performed (register postmodification is performed if specified). The Y operand can specify a word memory location or an accumulator. Based on dauc[6], the Z-field operation can be conditional. If dauc[6] = 0, the Z-field operation, if specified, is not conditional. Note that if the test fails, the previous value of aN is stored to memory. If dauc[6] = 1, the write to memory, if specified, is also conditional, but the address postmodifications are always performed. If written, the value of aN is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ifalt).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r11-- = a2 = ifalt(*r7--)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 1 0 1	1 0	0 0 0 0 0 0 0	0 1 1 1 1 1 0	1 0 1 1 1 1 0
Meaning		ifalt	a2		*r7--	*r11--

INSTRUCTION: [if (COND)]rD1 = {[(long)],(short)} rS {+,–} 1 (1)
[if (COND)] sp = [(long)] sp {++,--} (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

(1) This instruction increments or decrements the contents of register rS by 1 and places the result in register rD1 (can not be the stack pointer). The items in the brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) operation, the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

(2) This instruction increments or decrements the contents of the register sp (stack pointer) by 4. The source register and the destination register are sp. The instruction can only be executed as 32-bits. **This instruction disables interrupts for one instruction cycle**. If the instruction is conditionally executed and the COND is true, the result is placed in the register sp, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register sp nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD1	r0—r20,r22	r0 (hard-wired zero), or one of 21 CAU registers
rS	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers

INSTRUCTION FORMAT (1-2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
F Field: Specifies the ALU function to be performed (+).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(vs) r5 = r5 +1

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	0	0 0 1 1 0 0	0 0 0 0	0 0 1 0 1	0 0 1 0 1	0 0 0 1 1 1	1 0 1 1 1
Meaning	(short)		+	r5	r5	vs	(+1)

INT16

INSTRUCTION: [Z =] aN = int16(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction converts the floating-point operand Y to a 16-bit 2's complement integer. The conversion can be done using one of three modes specified by the dauc register bits 4 and 5: round to nearest (dauc[5—4] = 00), truncate towards -∞ (dauc[5—4] = 10), or truncate towards zero (dauc[5—4] = 11). The default (on reset) value of dauc[5—4] = 00. The Y operand can specify an accumulator or a float memory location. The result is stored in the upper 16 bits of an accumulator (aN). The lower portion of aN contains unpredictable data. If the Y operand exceeds the range of allowable 16-bit integers (-2^{15}, $2^{15} - 1$), the result is a saturated 16-bit integer. If a Z field is specified, the result is stored to a half-word address in memory. In order to write the 16-bit result to memory as a 16-bit write, the Z-write must be performed in this instruction. When the post-increment or the post-decrement addressing mode is used for the Z operand, the value of the modifier is +2 or -2, respectively.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	16-bit memory location. ++ is +2 and -- is -2. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (int16).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r3 = a0 = int16(a0)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 0 1 1	0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 1 1 0 0 0
Meaning		int16	a0		a0	*r3

INSTRUCTION: [Z =] aN = int32(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction converts the floating-point operand Y to a 32-bit 2's complement integer. The conversion can be done using one of three modes specified by the dauc register bits 4 and 5: round to nearest (dauc[5—4] = 00), truncate towards -∞ (dauc[5—4] = 10), or truncate towards zero (dauc[5—4] = 11). The default (on reset) value of dauc[5—4] = 00. The Y operand can specify an accumulator or a 32-bit memory location. The result is stored in the 32 MSBs of an accumulator (aN), the mantissa, and guard bits. The remaining bits of aN contain unpredictable data. If the Y operand exceeds the range of allowable 32-bit integers (-2^{31} , $2^{31} - 1$), the result is a saturated 32-bit integer. If a Z field is specified, the result is stored to a word address in memory. In order to get the result correctly formatted in memory, the Z-write must be performed in this instruction.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (int32).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r12++ = a2 = int32(*r1++)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	1 0 1 1	1 0	0 0 0 0 0 0 0	0 0 0 1 1 1 1	1 1 0 0 1 1 1
Meaning		int32	a2		*r1++	*r12++

IRETURN

INSTRUCTION: ireturn

CLASS: CA-Control

CAU FLAGS AFFECTED: Restored from interrupt shadow register

DAU FLAGS AFFECTED: Restored from interrupt shadow register

DESCRIPTION

This instruction performs a return-from-interrupt operation. This operation consists of returning DSP3210 control to the point of interruption and restoring the internal registers from the shadow registers. This instruction is encoded and executes as *if(true) goto pcsh.*

LATENCY

Unlike other CA control group instructions, the ireturn instruction does not have to be followed by a nop, since the instruction following the ireturn in memory is not executed. The instruction that was fetched prior to the interrupt is executed in its place. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Formats 0b and 1b)

Bit	31—27	26—21	20—16	15—0
Field	1 0 0 0 0	C	rB	N

C Field: Specifies the condition (true).
rB Field: Specifies the base address register (pcsh).
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address (zero).

EXAMPLE (For field encodings refer to Chapter 10.)

ireturn

Bit	31—27	26—21	20—16	15—0
Encoding	1 0 0 0 0	0 0 0 0 0 1	1 1 1 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		true	pcsh	0x0000

INSTRUCTION:

rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]}	MEM	(1)
rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]}	*L	(2)
rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]}	iorS	(3)

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves data from a memory location or IO register specified on the right-hand side to a register rD. The default size of the transfer is (long) and does not have to be specified. The size of the transfer and the CAU register bits affected by the transfer are specified by the key words: (char), (byte), (hbyte), (short), (ushort), and (long). (char), (byte) and (hbyte) are 8-bit transfers. (char) loads bits 7—0 with the data and sign extends bit 7 to bit 31. (byte) loads bits 7—0 with the data and zero extends to bit 31. (hbyte) loads bits 15—8 with the data, and zeros bits 7—0 and 31—16. (short) and (ushort) are 16-bit transfers. (short) loads bits 15—0 with the data and sign extends to bit 31. (ushort) loads bits 15—0 with the data and zero extends to bit 31. (long) is a 32-bit transfer that loads bits 31—0 with the data (see below). **This instruction disables interrupts for one instruction cycle.**

(1) This instruction moves data from a memory location addressed using the register indirect addressing mode to the register rD. Data can be referenced as 8-, 16-, or 32-bits. Post-incrementation or post-decrementation is a function of the data size (8-bit, ±1; 16-bit, ±2; 32-bit, ±4).

(2) This instruction moves data from a memory location addressed using the direct addressing mode (*L) to the register rD. L is a 16-bit unsigned integer that is concatenated with the upper 16-bits representing the location of on-chip memory resources (0x0000 or 0x5003, based on pcw[10]) to form the address.

(3) This instruction moves data from an IO register to the register rD. The size of the selected IO register and the transfer size must be the same.

Destination CAU Register Bits Affected						
Type	**Size**	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	8	0	0	0	DATA	Zero-extend to upper 24-bits
(hbyte)	8	0	0	DATA	0	Zero-fill bits 7—0, zero-extend upper 16-bits
(char)	8	S	S	S	DATA	Sign-extend bit 7 to upper 24-bits
(short)	16	S	S	DATA	DATA	Sign-extend bit 15 to upper 16-bits
(ushort)	16	0	0	DATA	DATA	Zero-extend to upper 16-bits
(long)	32	DATA	DATA	DATA	DATA	Load bits 31—0.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location. ++ and -- are a function of the data size. rP is a memory pointer (rP = r0—r22); when using the sp (r21), the size of the transfer must be long. rI is an increment register (rI = r0—r22).
iorS	ior0—ior15	One of the IO registers.
L	16-bit number	Unsigned integer - used as a direct address

RESTRICTION

The CAU register loaded and the flags set as a result of the load cannot be referenced in the following instruction (See Section 4.4.1.3, Restriction 3 – CAU and IO Register Load).

INSTRUCTION (cont.): rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]} MEM
rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]} *L
rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]} iorS

INSTRUCTION FORMAT (1) (Format 7c)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	T	W	rH	rP	0	0 0 0 0 0	rI

INSTRUCTION FORMAT (2) (Format 7a)

Bit	31	30—26	25	24	23—21	20—16	15—0
Field	0	0 0 1 1 1	0	T	W	rH	L

INSTRUCTION FORMAT (3) (Format 7b)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	T	W	rH	0 0 0 0 0	1	0 0 0 0 0	iorH

T Field: Specifies the direction of the transfer (to a register).
W Field: Specifies the size of the transfer.
rH Field: Specifies the source/destination CAU register (destination).
rP Field: Specifies the memory pointer register.
rI Field: Specifies the increment register.
L Field: Specifies a 16-bit unsigned integer.
iorH Field: Specifies the source/destination IO register (source)

EXAMPLE (For field encodings refer to Chapter 10.)

r10 = (ushort) *0xaa

Bit	31	30—26	25	24	23—21	20—16	15—0
Encoding	0	0 0 1 1 1	0	0	0 1 0	0 1 0 1 0	0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0
Meaning				(to)	(ushort)	r10	0x00aa

INSTRUCTION: iorD = {(byte),(short),[(long)]} MEM

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves data from a memory location to an IO register. The default size of the transfer is (long) and does not have to be specified. The size of the transfer and the IO register bits affected by the transfer are specified by the key words: (byte), (short), and (long). (byte) is an 8-bit transfer that loads bits 7—0 with the data and does not affect bits 8 to bit 31. (short) is a 16-bit transfer that loads bits 15—0 with the data and does not affect bits 16 to 31. (long) is a 32-bit transfer that loads bits 31—0 with the data (see below). **This instruction disables interrupts for one instruction cycle.**

Destination IO Register Bits Affected						
Type	**Size**	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	8	X	X	X	DATA	Load bits 7—0 (X means not affected)
(short)	16	X	X	DATA	DATA	Load bits 15—0
(long)	32	DATA	DATA	DATA	DATA	Load bits 31—0

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location. ++ and -- are a function of the data size. rP is a memory pointer (rP = r0—r22); when using the sp (r21), the size of the transfer must be long. rI is an increment register (rI = r0—r22).
iorD	ior0—ior15	One of the IO registers

RESTRICTION

The IO register loaded cannot be referenced in the following instruction (See Section 4.4.1.3, Restriction 3 – CAU and IO Register Load). A dauc load from memory must be followed by a CA instruction.

INSTRUCTION FORMAT (Format 7d)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	1	T	W	iorH	rP	0	0 0 0 0 0	rI

T Field: Specifies the direction of the transfer (to a register).
W Field: Specifies the size of the transfer.
rP Field: Specifies the memory pointer register.
rI Field: Specifies the increment register.
iorH Field: Specifies the source/destination IO register (destination).

EXAMPLE (For field encodings refer to Chapter 10.)

ior8 = (short) *r2

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Encoding	1	0 0 1 1 1	1	0	0 1 1	0 1 0 0 0	0 0 0 1 0	0	0 0 0 0 0	0 0 0 0 0
Meaning				to	(short)	ior8	r2			

NOP

INSTRUCTION: nop

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction is a no-op instruction.This instruction causes execution of the following instruction to be delayed by one processor cycle. This is a special case of the *if(COND) goto* instruction in which the branch is never taken.

INSTRUCTION FORMAT (Formats 0b and 1b)

Bit	31—27	26—21	20—16	15—0
Field	1 0 0 0 0	C	rB	N

C Field: Specifies the condition (false).
rB Field: Specifies the base address register (r0).
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address (zero).

EXAMPLE (For field encodings refer to Chapter 10.)

nop

Bit	31—27	26—21	20—16	15—0
Encoding	1 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		false	r0	0x0000

INSTRUCTION: [Z =] aN = oc(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction converts the floating-point operand specified by the Y field to an 8-bit μ-law, A-law, or unsigned linear value. The data type is specified by dauc register bits 3 and 1: μ-law (dauc[3,1]=0,1), A-law (dauc[3,1] = 0,0), unsigned (dauc[3,1] = 1,x). Conversion to unsigned can be done using one of three modes specified by the dauc register bits 4 and 5: round to nearest (dauc[5—4] = 00), truncate towards -∞ (dauc[5—4] = 10), or truncate towards zero (dauc[5—4] = 11). The default (on reset) value of the dauc register is 0.The Y operand can be a float memory location or an accumulator. The result is stored in the most significant byte of an accumulator (aN). The lower portion of the accumulator is written with unpredictable data. If the value of Y is outside the range of allowable μ-law/A-law/8-bit unsigned numbers, the limiting μ-law/A-law/8-bit unsigned value is returned. For more information see 8.2.4 Type Conversions. No flags are set if this saturation occurs. If a Z field is specified, the result is stored to a byte address in memory. When the post-increment or post-decrement addressing mode is used for the Z operand, the value of the modifier is +1 or –1, respectively. In order to execute the store as 8-bits, the Z-write must be performed in this instruction.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	8-bit memory location. ++ is +1 and -- is -1. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (oc).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a1 = oc(a1)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 0 0 1	0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 1 1 1
Meaning		oc	a1		a1	No write

OR

INSTRUCTION: rD = {[(long)],(short)} rD | N (1)
[if(COND)] rD = {[(long)],(short)} rS1 | rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz,v=c=0

DESCRIPTION

These instructions perform bit-by-bit, logical OR operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a bit-by-bit, logical OR operation on the contents of register rD with the value N and the result is placed in the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) This instruction performs a bit-by-bit, logical OR operation on the contents of register rS1 with the contents of register rS2, and the result is placed in the register rD. If the instruction is conditionally executed and the COND is true, the result of the operation on rS1 and rS2 is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (|).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r15 = (long) r15 | 0x55

Bit	31	30—25	24—21	20—16	15—0
Encoding	1	0 0 1 1 0 1	1 0 1 0	0 1 1 1 1	0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1
Meaning	(long)		|	r15	0x0055

INSTRUCTION: return (rM)

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves the value of rM into the program counter (pc). This is the special case of the unconditional goto instruction where only the CAU registers can be selected. Register rM holds the return address of a subroutine. This instruction is encoded and executes as *if (true)goto rM*. **This instruction disables interrupts for one instruction cycle.**

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rM	r1—r22	One of 22 CAU registers.

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Formats 0b and 1b)

Bit	31—27	26—21	20—16	15—0
Field	1 0 0 0 0	C	rB	N

C Field: Specifies the condition (true).
rB Field: Specifies the base address register (rM).
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address (zero).

EXAMPLE (For field encodings refer to Chapter 10.)

return (r18)

Bit	31—27	26—21	20—16	15—0
Encoding	1 0 0 0 0	0 0 0 0 0 1	1 0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		true	r18	0x0000

ROTATE LEFT

INSTRUCTION: [if (COND)] rD = {[(long)],(short)} rS <<<1

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

This instruction performs rotate left through carry, i.e., shifts the contents of register rS to the left by one bit and shifts the carry from the previous instruction into bit 0. The result is placed in the register rD. The items in the brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) operation, the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers

NSTRUCTION FORMAT (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS Field: Specifies the source register.
F Field: Specifies the ALU function to be performed (<<<).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(cs) r5 = r6 <<<1

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 0 1 1	0 0 1 0 1	0 0 1 1 0	0 0 1 0 0 1	0 0 1 1 0
Meaning	(long)		<<<	r5	r6	cs	r6

INSTRUCTION: [if (COND)] rD = {[(long)], (short)} rS >>>1

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzc, v=0

DESCRIPTION

This instruction performs rotate right through carry, i.e., shifts the contents of register rS to the right by one bit and shifts the carry from the previous instruction into the most significant bit (MSB). The result is placed in the register rD. The items in the brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) operation, the carry is shifted into bit 15, and the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers

NSTRUCTION FORMAT (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS Field: Specifies the source register.
F Field: Specifies the ALU function to be performed (>>>).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(cs) r5 = r6 >>>1

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 0 0 1	0 0 1 0 1	0 0 1 1 0	0 0 1 0 0 1	0 0 1 1 0
Meaning	(long)		>>>	r5	r6	cs	r6

ROUND

INSTRUCTION: [Z =] aN = round(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction rounds the floating-point operand Y to 32-bit floating-point format. It is intended to operate on the contents of an accumulator that contains eight extra bits of precision in the mantissa (although other operands are allowed). The 40-bit operand is rounded to the nearest 32-bit representation and the result is stored in an accumulator (aN). The eight guard bits in the accumulator aN are set to zero. If the 40-bit operand is exactly between two 32-bit values, it is rounded to the greater value. The Y operand can specify an accumulator or a 32-bit memory location. If a Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (round).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder nput via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r12++r16 = a3 = round(*r1--)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 1 0 0	1 1	0 0 0 0 0 0 0	0 0 0 1 1 1 0	1 1 0 0 0 1 0
Meaning		round	a3		*r1--	*r12++r16

INSTRUCTION: [Z =] aN = seed(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZU, V=0

DESCRIPTION

This instruction produces a seed (reciprocal of Y) with mantissa good to three bits of accuracy by inverting all bits of Y, except the sign bit. The floating-point operand Y can specify a float memory location or an accumulator. The result is stored in an accumulator (aN). If a Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--, *rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (seed).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a3 = seed(*r9++)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	1 1 1 0	1 1	0 0 0 0 0 0 0	1 0 0 1 1 1 1	0 0 0 0 1 1 1
Meaning		seed	a3		*r9++	No write

SET

INSTRUCTION: rD = (short) N

CLASS: CA-Data Move

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

This instruction moves the value N into the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits. Flags are set according to 16-bit flag rules (see Table 4-1). The operation actually performed is rD = r0 + N.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (Format 5a)

Bit Field	31—26	25—21	20—16	15—0
	000101	rD	rS	N

rD Field: Specifies the destination register.
rS Field: Specifies the source register (r0).
N Field: Specifies a 16-bit unsigned integer that is sign-extended to 32-bits.

EXAMPLE (For field encodings refer to Chapter 10.)

r2= (short) 0x0800

Bit Encoding Meaning	31—26	25—21	20—16	15—0
	000101	00010	00000	0000100000000000
		r2	(r0)	0x0800

INSTRUCTION: rD = (ushort24) M

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves the value M into register rD. M is a 24-bit unsigned integer that is zero extended to 32-bits.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
M	24-bit number	unsigned integer

INSTRUCTION FORMAT (Format 8b)

Bit	31—29	28—21	20—16	15—0
Field	1 1 0	NE	rD	N

rD Field: Specifies the destination register.
NE Field: Specifies the most significant 8-bits of the 24-bit integer. NE concatonated with N forms the 24-bit unsigned integer (NE || N).
N Field: Specifies the least significant 16-bits of the 24-bit unsigned integer.

EXAMPLE (For field encodings refer to Chapter 10.)

r2= (ushort24) 0x080000

Bit	31—29	28—21	20—16	15—0
Encoding	1 1 0	0 0 0 0 1 0 0 0	0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		0x08	r2	0x0000

SFTRST

INSTRUCTION: sftrst

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction performs a soft reset of the processor level. It is intended to be used only from the error level and it returns the processor to the base level. By resetting registers and branching to zero an emulation of reset can be accomplished. This instruction is encoded as spc=(byte)r0.

INSTRUCTION FORMAT (Format 7b) sftrst

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	1	0 0 0	0 0 0 0 0	0 0 0 0 0	1	0 0 0 0 0	0 1 0 1 0

INSTRUCTION:

rD = {[(long)], (short)} rD << N (1)
[if(COND)] rD = {[(long)], (short)} rS1 << rS2 (2)
[if(COND)] rD = {[(long)], (short)} rS << 1 (3)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform logical left shift operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. A (long) left shift can be considered an arithmetic left shift, but note that overflow is not detected. (short) indicates that a 16-bit operation is desired. The (short) left shift is a logical left shift and the upper 16 bits of the result are zero. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

(1) This instruction performs a logical left shift on the contents of register rD by N and the result is placed in the register rD. Only the least significant 5 bits of N are used to specify a shift of 0 to 31 binary places.

(2) This instruction performs a logical left shift on the contents of register rS1 by the contents of register rS2, and the result is placed in the register rD. Only the least significant 5 bits of rS2 are used to specify a shift of 0 to 31 binary places.

(3) This instruction performs a conditional logical left shift by one of the register rS and the result is placed in the register rD.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2-3) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (<<).
C Field: Specifies the condition being tested.

SHIFT LEFT

INSTRUCTION (cont.): rD = {[(long)], (short)} rD << N
[if(COND)] rD = {[(long)], (short)} rS1 << rS2
[if(COND)] rD = {[(long)], (short)} rS << 1

EXAMPLE (For field encodings refer to Chapter 10.)

if(ane) r3 = (short) r2 << 1

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	0	0 0 1 1 0 0	1 1 0 0	0 0 0 1 1	0 0 0 1 0	0 1 0 1 0 0	1 0 1 1 1
Meaning	(short)		<<	r3	r2	ane	(1)

INSTRUCTION: rD = [(long)] rS <<| N

CLASS: CA-Arithmetic Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

This instruction is a shift and OR instruction. The value N is logically shifted to the left by 16 binary places and OR'ed with the contents of rS and the result is stored in the register rD. The instruction is always executed as 32-bits, (long). Specification of (long) is optional. Flags are set according to 32-bit flag rules.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (Format 4b)

Bit	31—26	25—21	20—16	15—0
Field	1 0 0 1 0 0	rD	rS	N

rD Field: Specifies the destination register.
rS Field: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer.

EXAMPLE (For field encodings refer to Chapter 10.)

r2= (long) r1 <<| 0x0800

Bit	31—26	25—21	20—16	15—0
Encoding	1 0 0 1 0 0	0 0 0 1 0	0 0 0 0 1	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
Meaning		r2	r1	0x0800

SHIFT RIGHT

INSTRUCTION: rD = {[(long)], (short)} rD >> N (1)
[if(COND)] rD = {[(long)], (short)} rS1 >> rS2 (2)
[if(COND)] rD = {[(long)], (short)} rS >> 1 (3)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform logical right shift operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) logical right shift, the upper 16 bits of the result are zero. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

(1) This instruction performs a logical right shift on the contents of register rD by N and the result is placed in the register rD. Only the least significant 5 bits of N are used to specify a shift of 0 to 31 binary places.

(2) This instruction performs a logical right shift on the contents of register rS1 by the contents of register rS2, and the result is placed in the register rD. Only the least significant 5 bits of rS2 are used to specify a shift of 0 to 31 binary places.

(3) This instruction performs a conditional logical right shift by one of the register rS and the result is placed in the register rD.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2-3) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (>>).
C Field: Specifies the condition being tested.

INSTRUCTION (cont.): rD = {[(long)], (short)} rD >> N
[if(COND)] rD = {[(long)], (short)} rS1 >> rS2
[if(COND)] rD = {[(long)], (short)} rS >> 1

EXAMPLE (For field encodings refer to Chapter 10.)

r2 = (long) r2 >> 30

Bit	31	30—25	24—21	20—16	15—0
Encoding	1	0 0 1 1 0 1	1 1 0 0	0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0
Meaning	(long)		>>	r2	30

SHIFT RIGHT-ARITH

INSTRUCTION: rD = {[(long)], (short)} rD $>> N
[if(COND)] rD = {[(long)], (short)} rS1 $>> rS2
[if(COND)] rD = {[(long)], (short)} rS / 2

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform arithmetic right shift operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) arithmetic right shift, the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

(1) This instruction performs an arithmetic right shift on the contents of register rD by N and the result is placed in the register rD. Only the least significant 5 bits of N are used to specify a shift of 0 to 31 binary places.

(2) This instruction performs an arithmetic right shift on the contents of register rS1 by the contents of register rS2, and the result is placed in the register rD. Only the least significant 5 bits of rS2 are used to specify a shift of 0 to 31 binary places.

(3) This instruction performs a divide-by-two operation; i.e., the arithmetic right shift with sign extension. The contents of register rS are divided by two, and the result is placed in register rD.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2-3) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed ($>>).
C Field: Specifies the condition being tested.

INSTRUCTION (cont.): rD = {[(long)], (short)} rD $>> N
[if(COND)] rD = {[(long)], (short)} rS1 $>> rS2
[if(COND)] rD = {[(long)], (short)} rS /2

EXAMPLE (For field encodings refer to Chapter 10.)

r4 = (long) r2 $>> r13

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 1 0 1	0 0 1 0 0	0 0 0 1 0	0 0 0 0 0 1	0 1 1 0 1
Meaning	(long)		$>>	r4	r2	true	r13

STORE

INSTRUCTION:

MEM = {(byte), (hbyte), (short), [(long)]} rS (1)
*L = {(byte), (hbyte), (short), [(long)]} rS (2)
iorD = {(byte), (hbyte), (short), [(long)]} rS (3)

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves data from the register rS to a memory location or IO register specified on the left-hand side. The default size of the transfer is (long) and does not have to be specified. The CAU register bits selected and the size of the transfer are specified by the key words: (byte), (hbyte), (short), and (long). (byte) and (hbyte) are 8-bit transfers that select register bits 7—0 and 15—8 respectively. (short) is a 16-bit transfer that selects register bits 15—0. (long) is a 32-bit transfer that selects registers 31—0 (see below).

(1) This instruction moves data from the register rS to a memory location addressed using the register indirect addressing mode. Data can be written as 8-, 16-, or 32-bits. Post-incrementation and post-decrementation is a function of the data size (8-bit, ±1; 16-bit, ±3; 32-bit, ±4).

(2) This instruction moves data from the register rS to a memory location addressed using the direct addressing mode (*L). L is a 16-bit unsigned integer that is concatenated with the upper 16-bits representing the location of on-chip memory resources (0x0000 or 0x5003, based on pcw[10]) to form the address.

(3) This instruction moves data from the register rS to an IO register. The transfer size and the size of the selected IO register must be the same.

Source CAU Register Bits Selected						
Type	**Size**	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	8	X	X	X	DATA	Store bits 7—0
(hbyte)	8	X	X	DATA	X	Store bits 15—8
(short)	16	X	X	DATA	DATA	Store bits 15—0
(long)	32	DATA	DATA	DATA	DATA	Store bits 31—0

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rS	r0—r22, pc, pcsh	r0 (hard-wired zero), one of 22 CAU registers, the program counter, or the program counter shadow register
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location. ++ is +1, +2, or +4 for data transfer sizes of 8-, 16-, and 32-bits. -- is -1, -2, and –4 for data transfer sizes of 8-, 16-, and 32-bits. rP is a memory pointer (rP = r0—r22); when using the sp (r21), the size of the transfer must be long. rI is an increment register (rI = r0—r22).
iorD	ior0—ior15	One of the IO registers
L	16-bit number	Unsigned integer - used as a direct address

RESTRICTION

A CAU or IO register store instruction cannot immediately follow a DA instruction (multiply/accumulate or special function) that references memory in the Y field (see Section 4.4.1.2, Restriction 2 – CAU and IO Register Store).

INSTRUCTION (cont.): MEM = {(byte), (hbyte), (short), [(long)]} rS
*L = {(byte), (hbyte), (short), [(long)]} rS
iorD = {(byte), (hbyte), (short), [(long)]} rS

INSTRUCTION FORMAT (1) (Format 7c)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	T	W	rH	rP	0	0 0 0 0 0	rI

INSTRUCTION FORMAT (2) (Format 7a)

Bit	31	30—26	25	24	23—21	20—16	15—0
Field	0	0 0 1 1 1	0	T	W	rH	L

INSTRUCTION FORMAT (3) (Format 7b)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	T	W	rH	0 0 0 0 0	1	0 0 0 0 0	iorH

T Field: Specifies the direction of the transfer (to).
W Field: Specifies the size of the transfer.
rH Field: Specifies the source/destination register (source).
rP Field: Specifies the memory pointer register.
rI Field: Specifies the increment register.
iorH Field: Specifies the source/destination IO register (destination).
L Field: Specifies a 16-bit unsigned integer.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++ = (hbyte) r3

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Encoding	1	0 0 1 1 1	0	1	1 0 0	0 0 0 1 1	0 0 0 0 1	0	0 0 0 0 0	1 0 1 1 1
Meaning				from	(hbyte)	r3	r1			rI

STORE-IOR

INSTRUCTION: MEM = {(byte), (short), [(long)]} iorS

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves data from an IO register to a memory location. The default size of the transfer is (long) and does not have to be specified. The size of the transfer and the IO register bits selected by the transfer are specified by the key words: (byte), (short), and (long). (byte) is an 8-bit transfer that selects bits 7—0. (short) is a 16-bit transfer that selects bits 15—0 . (long) is a 32-bit transfer that selects bits 31—0 (see below). Post-incrementation and post-decrementation is a function of the data size (8-bit, ±1; 16-bit, ±2, ; 32-bit, ±4).

Source IO Register Bits Selected						
Type	**Size**	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	8	X	X	X	DATA	Select bits 7—0
(short)	16	X	X	DATA	DATA	Select bits 15—0
(long)	32	DATA	DATA	DATA	DATA	Select bits 31—0

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location. ++ is +1, +2, or +4 for data transfer sizes of 8-, 16-, and 32-bits. -- is -1, -2, and –4 for data transfer sizes of 8-, 16-, and 32-bits. rP is a memory pointer (rP = r0—r22); when using the sp (r21), the size of the transfer must be long. rI is an increment register (rI = r0—r22).
iorS	ior0—ior15	One of the IO registers

RESTRICTION

A CAU or IO register store instruction cannot immediately follow a DA instruction (multiply/accumulate or special function) that references memory in the Y field (see Section 4.4.1.2, Restriction 2 – CAU and IO Register Store).

INSTRUCTION FORMAT (Format 7d)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	1	T	W	iorH	rP	0	0 0 0 0 0	rI

T Field: Specifies the direction of the transfer (from a register).
W Field: Specifies the size of the transfer.
rP Field: Specifies the memory pointer register.
rI Field: Specifies the increment register.
iorH Field: Specifies the source/destination IO register (source).

EXAMPLE (For field encodings refer to Chapter 10.)

*r1 = (short) ior0

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Encoding	1	0 0 1 1 1	1	1	1 1 1	0 0 0 0 0	0 0 0 0 1	0	0 0 0 0 0	0 0 0 0 0
Meaning				from	(long)	ior0	r1			

INSTRUCTION:

rD = {[(long)], (short)} rD - N (1)
rD = {[(long)], (short)} N - rD (2)
[if(COND)] rD = {[(long)], (short)} rS1 - rS2 (3)
[if(COND)] rD = {[(long)], (short)} - rS (4)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

These instructions perform subtract operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. A (short) operation sign extends the result from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-32).

(1) is a right subtract instruction. This instruction subtracts the value N from the contents of register rD and places the result in the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) is a left subtract instruction. This instruction subtracts the contents of register rD from a value N and places the result in the register rD. N is a 2's complement number that is MSB-extended to 32 bits.

(3) is a conditional triadic subtract instruction. This instruction subtracts the contents of register rS2 from register rS1 and places the result in rD.

(4) conditionally changes the sign of the contents of register rS. The result is placed in register rD. This negate instruction sets overflow flag v to one (v = 1) for rS = 0xffff8000 and sets carry flag c to zero (c = 0) for rS = 0x00000000; in all other cases v = 0, c = 1.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

SUBTRACT

INSTRUCTION (cont.): rD = {[(long)], (short)} rD - N
rD = {[(long)], (short)} N - rD
[if(COND)] rD = {[(long)], (short)} rS1 - rS2
[if(COND)] rD = {[(long)], (short)} - rS

INSTRUCTION FORMAT (1-2) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (3–4) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (-).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(mi) r10 = -r5

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	0 1 0 0	0 1 0 1 0	0 0 0 0 0	0 0 0 0 1 1	0 0 1 0 1
Meaning	(long)		—	r10	r0	mi	r5

INSTRUCTION: waiti

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction initiates the wait-for-interrupt mode. The wait-for-interrupt mode is a power-saving mode that stops operations in the CAU and DAU until an interrupt is recognized. If an interrupt is pending when the *waiti* instruction is executed, the *waiti* instruction is ignored. A *nop* instruction should always follow the *waiti* instruction because when the interrupt is recognized this instruction is executed before the interrupt is taken. It should only be executed from the base level. This instruction is encoded as spc=(long)r0.

INSTRUCTION FORMAT (Format 7b) waiti

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	1	1 1 1	0 0 0 0 0	0 0 0 0 0	1	0 0 0 0 0	0 1 0 1 0