Chapter 13
Instruction Set

This chapter describes each instruction in the DSP56300 (family) core instruction set in detail. Instructions that allow parallel moves are so noted in both the **Operation** and the **Assembler Syntax** fields. The MOVE instruction is equivalent to a NOP with parallel moves, so a description of each parallel move accompanies the MOVE instruction details. When an instruction uses an accumulator as both a destination operand for Data ALU operation and a source for a parallel move operation, the parallel move operation uses the value in the accumulator before any Data ALU operation executes. Use **Table 13-1** to locate the page number of an instruction.

Table 13-1. DSP56300 Instruction Summary

Table 13-1. DSP56300 Instruction Summary (Continued)

Instruction	Page	Instruction	Page
CMP Compare	**page 13-46**	**CMPM** Compare Magnitude	**page 13-48**
CMPU Compare Unsigned	**page 13-49**	**DEBUG** Enter Debug Mode	**page 13-50**
DEBUGcc Enter Debug Mode Conditionally	**page 13-51**	**DEC** Decrement by One	**page 13-52**
DIV Divide Iteration	**page 13-53**	**DO** Start Hardware Loop	**page 13-57**
DMAC Double (Multi) Precision Multiply Accumulate With Right Shift	**page 13-56**	**DOR** Start PC-Relative Hardware Loop	**page 13-62**
DO FOREVER Start Infinite Loop	**page 13-60**	**ENDDO** End Current DO Loop	**page 13-67**
DOR FOREVER Start PC-Relative Infinite Loop	**page 13-65**	**EXTRACT** Extract Bit Field	**page 13-70**
EOR Logical Exclusive OR	**page 13-68**	**IFcc.U** Execute Conditionally With CCR Update	**page 13-74**
EXTRACTU Extract Unsigned Bit Field	**page 13-72**	**INC** Increment by One	**page 13-77**
ILLEGAL Illegal Instruction Interrupt	**page 13-76**	**Jcc** JumpConditionally	**page 13-80**
INSERT Insert Bit Field	**page 13-78**	**JMP** Jump	**page 13-83**
JCLR Jump if Bit Clear	**page 13-81**	**JSCLR** Jump to Subroutine if Bit Clear	**page 13-85**
JScc Jump to Subroutine Conditionally	**page 13-84**	**JSR** Jump to Subroutine	**page 13-89**
JSET Jump if Bit Set	**page 13-87**	**LRA** Load PC-Relative Address	**page 13-92**
JSSET Jump to Subroutine if Bit Set	**page 13-90**	**LSR** Logical Shift Right	**page 13-96**
LSL Logical Shift Left	**page 13-93**	**MAC** Signed Multiply Accumulate	**page 13-99**
LUA Load Updated Address	**page 13-98**	**MAC (su, uu)** Mixed Multiply Accumulate	**page 13-102**
MACI Signed Multiply Accumulate With Immediate Operand	**page 13-101**	**MACRI** Signed Multiply Accumulate and Round With Immediate Operand	**page 13-105**
MACR Signed Multiply Accumulate and Round	**page 13-103**	**MAXM** Transfer by Magniture	**page 13-107**
MAX Transfer by Signed Value	**page 13-106**	**MOVE** Move Data	**page 13-110**
MERGE Merge Two Half Words	**page 13-108**	No Parallel Data Move	**page 13-112**

Table 13-1. DSP56300 Instruction Summary (Continued)

Instruction	Page	Instruction	Page
R Register-to-Register Data Move	**page 13-115**	Immediate Short Data Move	**page 13-113**
X: X Memory Data Move	**page 13-118**	**U** Address Register Update	**page 13-117**
Y: Y Memory Data Move	**page 13-122**	**X:R** X Memory and Register Data Move	**page 13-120**
L: Long Memory Data Move	**page 13-126**	**R:Y** Register and Y Memory Data Move	**page 13-124**
MOVEC Move Control Register	**page 13-130**	**X:Y:** XY Memory Data Move	**page 13-128**
MOVEP Move Peripheral Data	**page 13-134**	**MOVEM** Move Program Memory	**page 13-132**
MPY (su, uu) Mixed Multiply	**page 13-139**	**MPY** Signed Multiply	**page 13-137**
MPYR Signed Multiply and Round	**page 13-141**	**MPYI** Signed Multiply With Immediate Operand	**page 13-140**
NEG Negate Accumulator	**page 13-144**	**MPYRI** Signed Multiply and Round With Immediate Operand	**page 13-143**
NORM Norm Accumulator Iteration	**page 13-147**	**NOP** No Operation	**page 13-145**
NOT Logical Complement	**page 13-149**	**NORMF** Fast Accumulator Normalization	**page 13-147**
ORI OR Immediate With Control Register	**page 13-152**	**OR** Logical Inclusive OR	**page 13-150**
PFLUSHUN Program cache Flush Unlocked Sectors	**page 13-154**	**PFLUSH** Program Cache Flush	**page 13-153**
PLOCKR Lock Instruction Cache Relative Sector	**page 13-157**	**PFREE** Program Cache Global Unlock	**page 13-155**
PUNLOCKR Unlock Instruction Cache Relative Sector	**page 13-159**	**PUNLOCK** Unlock Instruction Cache Sector	**page 13-158**
RESET Reset On-Chip Peripherals Devices	**page 13-162**	**REP** Repeat Next Instruction	**page 13-160**
ROL Rotate Left	**page 13-165**	**RND** Round Accumulator	**page 13-163**
RTI Return From Interrupt	**page 13-168**	**ROR** Rotate Right	**page 13-166**
SBC Subtract Long With Carry	**page 13-169**	**RTS** Return From Subroutine	**page 13-168**
SUB Subtract	**page 13-172**	**STOP** Stop Instruction Processing	**page 13-170**
SUBR Shift Right and Subtract Accumulators	**page 13-175**	**SUBL** Shift Left and Subtract Accumulators	**page 13-174**
Tcc Transfer Conditionally	**page 13-176**	**TFR** Transfer Data ALU Register	**page 13-178**

Table 13-1. DSP56300 Instruction Summary (Continued)

Instruction	Page	Instruction	Page
TRAP Software Interrupt	**page 13-179**	**TRAPcc** Conditional Software Interrupt	**page 13-180**
TST Test Accumulator	**page 13-181**	**VSL** Viterbi Shift Left	**page 13-182**
WAIT Wait for Interrupt or DMA Request	**page 13-183**		

ABS — Absolute Value — ABS

Operation

| D | → D (parallel move)

Assembler Syntax

ABS D (parallel move)

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Take the absolute value of the destination operand D and store the result in the destination accumulator.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

ABS D

23 — 16	15 — 8	7 — 0
Data Bus Move Field	0 0 1 0	d 1 1 0
Optinal Effective Address Extension		

Operation		Assembler Syntax	
S + C + D → D	(parallel move)	ADC S,D	(parallel move)

Instruction Fields

{S}	J	Source register [X,Y] (see **Table 12-13** on page 12-22)
{D}	d	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Add the source operand S and the Carry bit (C) of the Condition Code Register to the destination operand D and store the result in the destination accumulator. Long words (48 bits) can be added to the 56-bit destination accumulator. Note that the Carry bit is set correctly for multiple-precision arithmetic using long-word operands if the extension register of the destination accumulator (A2 or B2) is the sign extension of Bit 47 of the destination accumulator (A or B).

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	√
CCR							

√	Changed according to the standard definition.
—	Unchanged by the instruction.

Instruction Formats and opcodes

ADC S,D

23 16	15 8	7 0
Data Bus Move Field		0 0 1 J d 0 0 1
Optional Effective Address Extension		

Operation

S + D → D (parallel move)

#xx + D → D

#xxxx + D → D

Assembler Syntax

ADD S,D (parallel move)

ADD #xx,D

ADD #xxxx,D

Instruction Fields

{S}	JJJ	Source register [B/A,X,Y,X0,Y0,X1,Y1] (see **Table 12-13** on page 12-22)
{D}	d	Destination accumulator [A/B] (see **Table 12-13** on page 12-22)
{#xx}	iiiiii	6-bit Immediate Short Data
{#xxxx}		24-bit Immediate Long Data extension word

Description Add the source operand S to the destination operand D and store the result in the destination accumulator. The source can be a register (24-bit word, 48-bit long word, or 56-bit accumulator), 6-bit short immediate, or 24-bit long immediate. When 6-bit immediate data is used, the data is interpreted as an unsigned integer. That is, the six bits are right-aligned and the remaining bits are zeroed to form a 24-bit source operand. Note that the Carry bit(C) is set correctly using word or long-word source operands if the extension register of the destination accumulator (A2 or B2) is the sign extension of Bit 47 of the destination accumulator (A or B). Thus, the C bit is always set correctly using accumulator source operands, but it can be set incorrectly if A1, B1, A10, B10 or immediate operand are used as source operands and A2 and B2 are not replicas of Bit 47.

Condition Codes

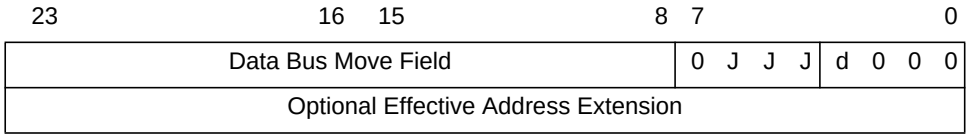
7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	√
CCR							

√ Changed according to the standard definition.

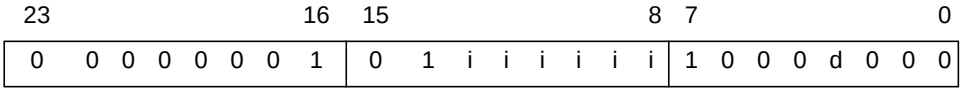
— Unchanged by the instruction.

Instruction Formats and opcodes

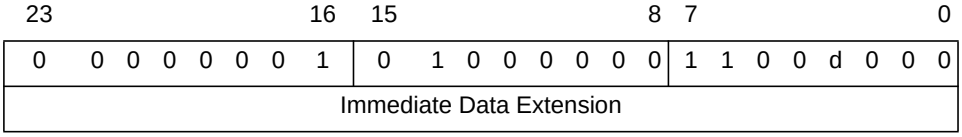
ADD S,D



ADD #xx,D



ADD #xxxx,D



ADDL Shift Left and Add Accumulators ADDL

Operation

$S + 2 * D \rightarrow D$ (parallel move)

Assembler Syntax

ADDL S,D (parallel move)

Instruction Fields

{D} d — Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

{S} The source accumulator is B if the destination accumulator (selected by the **d** bit in the opcode) is A, or A if the destination accumulator is B.

Description Add the source operand S to two times the destination operand D and store the result in the destination accumulator. The destination operand D is arithmetically shifted one bit to the left, and a 0 is shifted into the LSB of D prior to the addition operation. The Carry bit (C) is set correctly if the source operand does not overflow as a result of the left shift operation. The Overflow bit (V) may be set as a result of either the shifting or addition operation (or both). This instruction is useful for efficient divide and Decimation-In-Time (DIT) FFT algorithms.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	*	√
CCR							

V — Set if overflow has occurred in A or B result or the MSB of the destination operand is changed as a result of the instruction's left shift.
Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

ADDL S,D

23 — 16	15 — 8	7 — 0
Data Bus Move Field	0 0 0 1	d 0 1 0
Optional Effective Address Extension		

ADDR Shift Right and Add Accumulators ADDR

Operation

S + D / 2 → D (parallel move)

Assembler Syntax

ADDR S,D (parallel move)

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

{S} The source accumulator is B if the destination accumulator (selected by the **d** bit in the opcode) is A, or A if the destination accumulator is B.

Description Add the source operand S to one-half the destination operand D and store the result in the destination accumulator. The destination operand D is arithmetically shifted one bit to the right while the MS bit of D is held constant prior to the addition operation. In contrast to the ADDL instruction, the Carry bit (C) is always set correctly, and the Overflow bit (V) can only be set by the addition operation and not by an overflow due to the initial shifting operation. This instruction is useful for efficient divide and Decimation-In-Time (DIT) FFT algorithms.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	√
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

ADDR S,D

23 16	15 8	7 0
Data Bus Move Field	0 0 0 0	d 0 1 0
Optional Effective Address Extension		

AND — Logical AND — AND

Operation

S • D[47:24] → D[47:24] (parallel move)

#xx • D[47:24] → D[47:24]

#xxxx • D[47:24] → D[47:24]

where • denotes the logical AND operator

Assembler Syntax

AND S,D (parallel move)

AND #xx,D

AND #xxxx,D

Instruction Fields

{S}	**JJ**	Source input register [X0,X1,Y0,Y1] (see **Table 12-13** on page 12-22)
{D}	**d**	Destination accumulator [A/B] (see **Table 12-13** on page 12-22)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxx}		24-bit Immediate Long Data extension word

Description Logically AND the source operand S with bits 47–24 of the destination operand D and store the result in bits 47–24 of the destination accumulator. The source can be a 24-bit register, 6-bit short immediate, or 24-bit long immediate. This instruction is a 24-bit operation. The remaining bits of the destination operand D are not affected. When 6-bit immediate data is used, the data is interpreted as an unsigned integer. That is, the six bits are right aligned and the remaining bits are zeroed to form a 24-bit source operand.

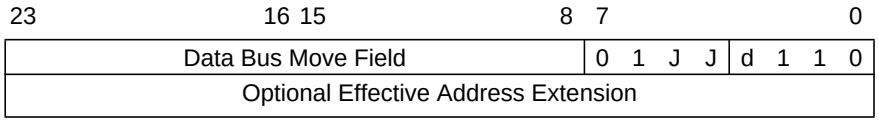
Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	—	—	—	*	*	*	—
CCR							

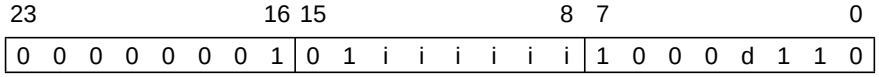
*	N	Set if bit 47 of the result is set.
*	Z	Set if bits 47-24 of the result are 0.
*	V	Always cleared.
√		Changed according to the standard definition.
—		Unchanged by the instruction.

Instruction Formats and opcodes

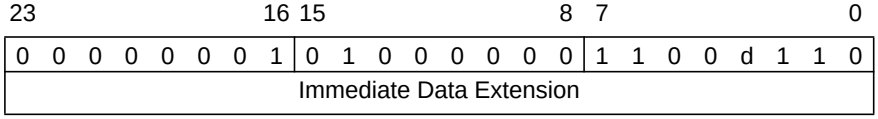
AND S,D



AND #xx,D



AND #xxxx,D



ANDI AND Immediate With Control Register ANDI

Operation
#xx • D → D
where • denotes the logical AND operator

Assembler Syntax
AND(I) #xx,D

Instruction Fields

{D}	**EE**	Program Controller register [MR,CCR,COM,EOM] (see **Table 12-13** on page 12-22)
{#xx}	**iiiiiiii**	Immediate Short Data

Description Logically AND the 8-bit immediate operand (#xx) with the contents of the destination control register D and store the result in the destination control register. The condition codes are affected only when the Condition Code Register (CCR) is specified as the destination operand.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

For CCR Operand

*	S	Cleared if Bit 7 of the immediate operand is cleared.
*	L	Cleared if Bit 6 of the immediate operand is cleared.
*	E	Cleared if Bit 5 of the immediate operand is cleared.
*	U	Cleared if Bit 4 of the immediate operand is cleared.
*	N	Cleared if Bit 3 of the immediate operand is cleared.
*	Z	Cleared if Bit 2 of the immediate operand is cleared.
*	V	Cleared if Bit 1 of the immediate operand is cleared.
*	C	Cleared if Bit 0 of the immediate operand is cleared.

For MR and OMR Operands

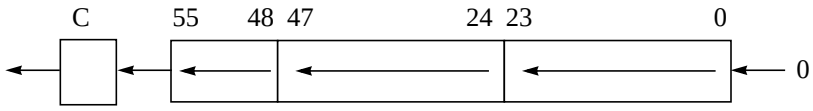
The condition codes are not affected using these operands.

Instruction Formats and opcodes

	23							16	15							8	7							0
AND(I) #xx,D	0	0	0	0	0	0	0	0	i	i	i	i	i	i	i	i	1	0	1	1	1	0	E	E

ASL Arithmetic Shift Accumulator Left ASL

Operation



Assembler Syntax

```
ASL D (parallel move)
ASL D #ii,S2,D
ASL S1,S2,D
```

Instruction Fields

{S2}	**S**	Source accumulator [A,B] ()
{D}	**D**	Destination accumulator [A,B] ()
{S1}	**sss**	Control register [X0,X1,Y0,Y1,A1,B1]
{#ii}	**iiiiii**	6-bit unsigned integer [0–40] denoting the shift amount

See **Table 12-13** on page 12-22

In the control register S1: bits 5–0 (LSB) are used as the #ii field, and the rest of the register is ignored.

Description

- *Single bit shift:* Arithmetically shift the destination accumulator D one bit to the left and store the result in the destination accumulator. The MSB of D prior to instruction execution is shifted into the Carry bit (C) and a 0 is shifted into the LSB of the destination accumulator D.
- *Multi-bit shift:* The contents of the source accumulator S2 are shifted left #ii bits. Bits shifted out of position 55 are lost except for the last bit, which is latched in the C bit. The vacated positions on the right are zero-filled. The result is placed into destination accumulator D. The number of bits to shift is determined by the 6-bit immediate field in the instruction, or by the 6-bit unsigned integer located in the six LSBs of the control register S1. If a zero shift count is specified, the C bit is cleared. The difference between ASL and LSL is that ASL operates on the entire 56 bits of the accumulator, and therefore, sets the Overflow bit (V) if the number overflows.

This is a 56-bit operation.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	*	*
CCR							

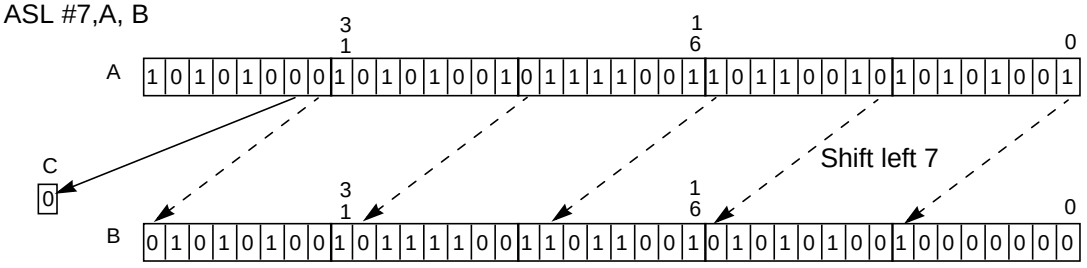
V — Set if Bit 55 is changed any time during the shift operation, cleared otherwise.

C — Set if the last bit shifted out of the operand is set, cleared for a shift count of 0, and cleared otherwise.

Changed according to the standard definition.

Example

ASL #7,A, B



Instruction Formats and opcodes

ASL D

23								8	7							0
Data Bus Move Field									0	0	1	1	d	0	1	0
Optional Effective Address Extension																

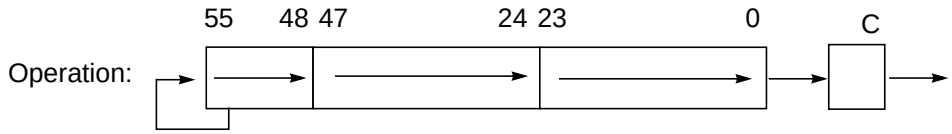
ASL #ii,S2,D

23								16	15							8	7							0
0	0	0	0	1	1	0	0		0	0	0	1	1	1	0	1	S	i	i	i	i	i	i	D

ASL S1,S2,D

23								16	15							8	7							0
0	0	0	0	1	1	0	0		0	0	0	1	1	1	1	0	0	1	0	S	s	s	s	D

ASR Arithmetic Shift Accumulator Right ASR



Assembler Syntax

```
ASR D (parallel move)
ASR D #ii, S2,D
ASR S1,S2,D
```

Instruction Fields

{S2}	**S**	Source accumulator [A,B]	
{D}	**D**	Destination accumulator [A,B]	See **Table 12-13** on page 12-22
{S1}	**sss**	Control register [X0,X1,Y0,Y1,A1,B1]	
{#ii}	**iiiiii**	6-bit unsigned integer [0-40] denoting the shift amount	

In the control register S1: bits 5-0 (LSB) are used as the #ii field, and the rest of the register is ignored.

Description

- *Single bit shift:* Arithmetically shift the destination operand D one bit to the right and store the result in the destination accumulator. The LSB of D prior to instruction execution is shifted into the Carry bit (C), and the MSB of D is held constant.
- *Multi-bit shift:* The contents of the source accumulator S2 are shifted right #ii bits. Bits shifted out of position 0 are lost except for the last bit, which is latched in the C bit. Copies of the MSB are supplied to the vacated positions on the left. The result is placed into destination accumulator D. The number of bits to shift is determined by the 6-bit immediate field in the instruction, or by the 6-bit unsigned integer located in the six 6 LSBs of the control register S1. If a zero shift count is specified, the C bit is cleared.

This is a 56- or 40-bit operation, depending on the SA bit value in the SR.

Note: If the number of shifts indicated by the 6 LSBs of the control register or by the immediate field exceeds the value of 55 (40 in Sixteen Bit Arithmetic mode), then the result is undefined.

Condition Codes

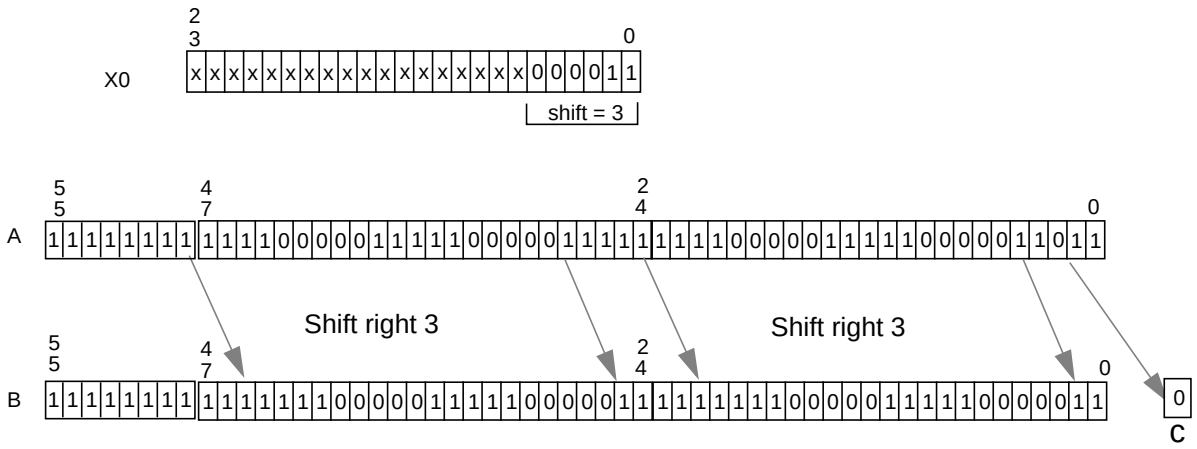
7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	*	*
CCR							

V This bit is always cleared.

C This bit is set if the last bit shifted out of the operand is set, cleared for a shift count of 0, and cleared otherwise.
Changed according to the standard definition.

Example

ASR X0,A,B



Instruction Formats and opcodes

ASR D

23								8	7							0
Data Bus Move Field									0	0	1	0	d	0	1	0
Optional Effective Address Extension																

ASR #ii,S2,D

23								16	15							8	7							0
0	0	0	0	1	1	0	0		0	0	0	1	1	1	0	0	S	i	i	i	i	i	i	D

ASR S1,S2,D

23								16	15							8	7							0
0	0	0	0	1	1	0	0		0	0	0	1	1	1	1	0	0	1	1	S	s	s	s	D

Bcc — Branch Conditionally — Bcc

Operation

If cc, then PC + xxxx → PC
else PC + 1 → PC

If cc, then PC + xxx → PC
else PC + 1 → PC

If cc, then PC + Rn → PC
else PC + 1 → PC

Assembler Syntax

Bcc xxxx

Bcc xxx

Bcc Rn

Instruction Fields

{cc}	**CCCC**	Condition code (see **Table 12-13** on page 12-22)
(xxxx)		24-bit PC Relative Long Displacement
{xxx}	**aaaaaaaaa**	Signed PC Relative Short Displacement
{Rn}	**RRR**	Address register [R0 – R7]

Description If the specified condition is true, program execution continues at location PC + displacement. If the specified condition is false, the PC is incremented and program execution continues sequentially. The displacement is a two's-complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Short Displacement and Address Register PC Relative addressing modes can be used. The Short Displacement 9-bit data is sign-extended to form the PC relative displacement. The conditions that the term "cc" can specify are listed on **Table 12-17** on page 12-28.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

		23 16	15 8	7 0
Bcc	xxxx	0 0 0 0 0 1 0 1	C C C C 0 1 a a	a a 0 a a a a a
			PC Relative Placement	
Bcc	xxx	0 0 0 0 0 1 0 1	C C C C 0 1 a a	a a 0 a a a a a
Bcc	Rn	0 0 0 0 1 1 0 1	0 0 0 1 1 R R R	0 1 0 0 C C C C

BCHG Bit Test and Change BCHG

Operation

D[n] → C	$\overline{D[n]} \rightarrow D[n]$
D[n] fi C	$\overline{D[n]} \rightarrow D[n]$
D[n] → C	$\overline{D[n]} \rightarrow D[n]$
D[n] → C	$\overline{D[n]} \rightarrow D[n]$
D[n] → C	$\overline{D[n]} \rightarrow D[n]$

Assembler Syntax

BCHG	#n,[XorY]:ea
BCHG	#n,[XorY]:aa
BCHG	#n,[XorY]:pp
BCHG	#n,[XorY]:qq
BCHG	#n,D

Instruction Fields

{#n}	**bbbb**	Bit number [0–23]
{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)
{X /Y}	**S**	Memory Space [X,Y] (see **Table 12-13** on page 12-22)
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]
{D}	**DDDDDD**	Destination register [all on-chip registers] (see **Table 12-13** on page 12-22)

Description Test the n^{th} bit of the destination operand D, complement it, and store the result in the destination location. The state of the n^{th} bit is stored in the Carry bit (C) of the CCR register. The bit to be tested is selected by an immediate bit number from 0 – 23. This instruction performs a read-modify-write operation on the destination location using two destination accesses before releasing the bus. This instruction provides a test-and-change capability, which is useful for synchronizing multiple processors using a shared memory. This instruction can use all memory alterable addressing modes.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

CCR Condition Codes

For destination operand SR:

* C Complemented if bit 0 is specified, unaffected otherwise.
* V Complemented if bit 1 is specified, unaffected otherwise.
* Z Complemented if bit 2 is specified, unaffected otherwise.
* N Complemented if bit 3 is specified, unaffected otherwise.
* U Complemented if bit 4 is specified, unaffected otherwise.
* E Complemented if bit 5 is specified, unaffected otherwise.
* L Complemented if bit 6 is specified, unaffected otherwise.
* S Complemented if bit 7 is specified, unaffected otherwise.

For other destination operands:

* C Set if bit tested is set, and cleared otherwise.
* V Not affected.
* Z Not affected.
* N Not affected.
* U Not affected.
* E Not affected.
* L Set according to the standard definition.
* S Set according to the standard definition.

MR Status Bits

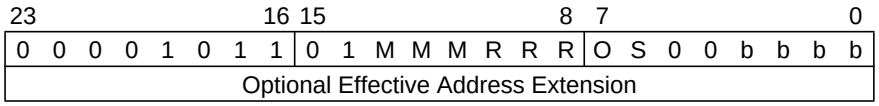
For destination operand SR:

* I0 Changed if bit 8 is specified, unaffected otherwise.
* I1 Changed if bit 9 is specified, unaffected otherwise.
* S0 Changed if bit 10 is specified, unaffected otherwise.
* S1 Changed if bit 11 is specified, unaffected otherwise.
* FV Changed if bit 12 is specified, unaffected otherwise.
* SM Changed if bit 13 is specified, unaffected otherwise.
* RM Changed if bit 14 is specified, unaffected otherwise.
* LF Changed if bit 15 is specified, unaffected otherwise.

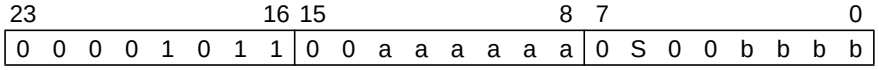
For other destination operands: MR status bits are not affected.

Instruction Formats and opcodes

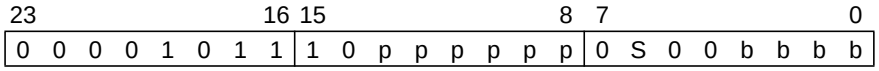
BCHG #n,[X or Y]:ea



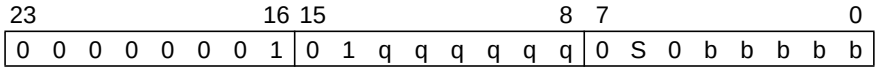
BCHG #n,[X or Y]:aa



BCHG #n,[X or Y]:pp



BCHG #n,[X or Y]:qq



BCHG #n,D

Operation		Assembler Syntax	
D[n] → C	0 → D[n]	BCLR	#n,[XorY]:ea
D[n] → C	0 → D[n]	BCLR	#n,[XorY]:aa
D[n] → C	0 → D[n]	BCLR	#n,[XorY]:pp
D[n] → C	0 → D[n]	BCLR	#n,[XorY]:qq
D[n] → C	0 → D[n]	BCLR	#n,D

Instruction Fields

{#n}	**bbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address
{X/Y}	**S**	Memory Space [X,Y]
{aa}	**aaaaaa**	Absolute Address [0–63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]
{D}	**DDDDDD**	Destination register [all on-chip registers]

See **Table 12-13** on page 12-22

Description Test the n^{th} bit of the destination operand D, clear it and store the result in the destination location. The state of the n^{th} bit is stored in the Carry bit (C) of the CCR register. The bit to be tested is selected by an immediate bit number from 0–23. This instruction performs a read-modify-write operation on the destination location using two destination accesses before releasing the bus. This instruction provides a test-and-clear capability, which is useful for synchronizing multiple processors using a shared memory. This instruction can use all memory alterable addressing modes.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

CCR Condition Codes

For destination operand SR:

* C Cleared if bit 0 is specified, unaffected otherwise.
* V Cleared if bit 1 is specified, unaffected otherwise.
* Z Cleared if bit 2 is specified, unaffected otherwise.
* N Cleared if bit 3 is specified, unaffected otherwise.
* U Cleared if bit 4 is specified, unaffected otherwise.
* E Cleared if bit 5 is specified, unaffected otherwise.
* L Cleared if bit 6 is specified, unaffected otherwise.
* S Cleared if bit 7 is specified, unaffected otherwise.

For other destination operands:

* C This bit is set if bit tested is set, and cleared otherwise.
* V Unaffected.
* Z Unaffected.
* N Unaffected.
* U Unaffected.
* E Unaffected.
* L This bit is set according to the standard definition.
* S This bit is set according to the standard definition.

MR Status Bits

For destination operand SR:

* I0 Changed if bit 8 is specified, unaffected otherwise.
* I1 Changed if bit 9 is specified, unaffected otherwise.
* S0 Changed if bit 10 is specified, unaffected otherwise.
* S1 Changed if bit 11 is specified, unaffected otherwise.
* FV Changed if bit 12 is specified, unaffected otherwise.
* SM Changed if bit 13 is specified, unaffected otherwise.
* RM Changed if bit 14 is specified, unaffected otherwise.
* LF Changed if bit 15 is specified, unaffected otherwise.

Instruction Formats and opcodes

BCLR #n,[X or Y]:ea

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	M	M	M	R	R	R	0	S	0	0	b	b	b	b
Optional Effective Address Extension																							

BCLR #n,[X or Y]:aa

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	a	a	a	a	a	a	0	S	0	0	b	b	b	b

BCLR #n,[X or Y]:pp

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	0	p	p	p	p	p	p	0	S	0	0	b	b	b	b

BCLR #n,[X or Y]:qq

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	q	q	q	q	q	q	0	S	0	0	b	b	b	b

BCLR #n,D

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	1	D	D	D	D	D	D	0	1	0	0	b	b	b	b

BRA — Branch Always — BRA

Operation

PC + xxxx → Pc

PC + xxx → Pc

PC + Rn → Pc

Assembler Syntax

BRA xxxx

BRA xxx

BRA Rn

Instruction Fields

{xxxx}		24-bit PC-Relative Long Displacement
{xxx}	**aaaaaaaaa**	Signed PC-Relative Short Displacement
{Rn}	**RRR**	Address register [R0–R7]

Description Program execution continues at location PC + displacement. The displacement is a two's-complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Short Displacement and Address Register PC Relative addressing modes may be used. The Short Displacement 9-bit data is sign-extended to form the PC relative displacement.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

BRA xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0
PC-Relative Displacement																							

BRA xxx

23							16	15							8	7							0
0	0	0	0	0	1	0	1	0	0	0	0	1	1	a	a	a	a	0	a	a	a	a	a

BRA Rn

23							16	15							8	7							0
0	0	0	0	1	1	0	1	0	0	0	1	1	R	R	R	1	1	0	0	0	0	0	0

BRCLR Branch if Bit Clear BRCLR

Operation

If	S{n}=0	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC
If	S{n}=0	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC
If	S{n}=0	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC
If	S{n}=0	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC
If	S{n}=0	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC

Assembler Syntax

BRCLR	#n,[X or Y]:ea,xxxx
BRCLR	#n,[X or Y],aa,xxxx
BRCLR	#n,[X or Y]:pp,xxxx
BRCLR	#n,[X or Y]:qq,xxxx
BRCLR	#n,S,xxxx

Instruction Fields

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address
{X/Y}	**S**	Memory Space [X,Y]
{xxxx}		24-bit PC relative displacement
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers])

See **Table 12-13** on page 12-22

Description The nth bit in the source operand is tested. If the tested bit is cleared, program execution continues at location PC+displacement. If the tested bit is set, the PC is incremented and program execution continues sequentially. However, the address register specified in the effective address field is always updated independently of the condition. The displacement is a 2's complement 24-bit integer that represents the relative distance from the current PC to the destination PC. The 24-bit displacement is contained in the extension word of the instruction. All memory alterable addressing modes may be used to reference the source operand. Absolute Short, I/O Short and Register Direct addressing modes may also be used. Note that if the specified source operand S is the SSH, the stack pointer register will be decremented by one. The bit to be tested is selected by an immediate bit number 0-23.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition

— Unchanged by the instruction

Instruction Formats and opcodes

BRCLR #n,[X or Y]:ea,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	0	M	M	M	R	R	R	0	S	0	b	b	b	b	b
PC-Relative Displacement																							

BRCLR #n,[X or Y]:aa,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	0	a	a	a	a	a	a	1	S	0	b	b	b	b	b
PC-Relative Displacement																							

BRCLR #n,[X or Y]:pp,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	1	p	p	p	p	p	p	0	S	0	b	b	b	b	b
PC-Relative Displacement																							

BRCLR #n,[X or Y]:qq,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	0	q	q	q	q	q	q	0	S	0	b	b	b	b	b
PC-Relative Displacement																							

BRCLR #n,S,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	1	D	D	D	D	D	D	1	0	0	b	b	b	b	b
PC-Relative Displacement																							

BRKcc Exit Current DO Loop Conditionally BRKcc

Operation

If cc LA + 1→ PC; SSL(LF,FV) → SR; SP – 1 → SP
SSH → LA; SSL → LC; SP – 1 → SP
else PC + 1 → PC

Assembler Syntax

BRKcc

Instruction Fields

{cc} **CCCC** Condition code (see **Table 12-18** on page 12-28)

Description Exits conditionally the current hardware DO loop before the current Loop Counter (LC) equals 1. It also terminates the DO FOREVER loop. If the value of the current DO LC is needed, it must be read before the execution of the BRKcc instruction. Initially, the PC is updated from the LA, the Loop Flag (LF) and the Forever flag (FV) are restored and the remaining portion of the Status Register (SR) is purged from the system stack. The LA and the LC registers are then restored from the system stack. The conditions that the term “cc” can specify are listed in **Table 12-18** on page 12-28.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

	23							16	15							8	7							0
BRKcc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	C	C	C	C

BRSET Branch if Bit Set BRSET

Operation

If	S{n}=1	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC
If	S{n}=1	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC
If	S{n}=1	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC
If	S{n}=1	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC
If	S{n}=1	then	PC+xxxx	→	PC
		else	PC+ 1	→	PC

Assembler Syntax

BRSET	#n,[X or Y]:ea,xxxx
BRSET	#n,[X or Y],aa,xxxx
BRSET	#n,[X or Y]:pp,xxxx
BRSET	#n,[X or Y]:qq,xxxx
BRSET	#n,S,xxxx

Instruction Fields

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address
{X/Y}	**S**	Memory Space [X,Y])
{xxxx}		24-bit PC relative displacement
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers]

See **Table 12-13** on page 12-22

Description The nth bit in the source operand is tested. If the tested bit is set, program execution continues at location PC+displacement. If the tested bit is cleared, the PC is incremented and program execution continues sequentially. However, the address register specified in the effective address field is always updated independently of the condition. The displacement is a 2’s complement 24-bit integer that represents the relative distance from the current PC to the destination PC. The 24-bit displacement is contained in the extension word of the instruction. All memory alterable addressing modes may be used to reference the source operand. Absolute Short, I/O Short and Register Direct addressing modes may also be used. Note that if the specified source operand S is the SSH, the stack pointer register will be decremented by one. The bit to be tested is selected by an immediate bit number 0-23.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition

— Unchanged by the instruction

Instruction Formats and opcodes

BRSET #n,[X or Y]:ea,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	0	M	M	M	R	R	R	0	S	1	b	b	b	b	b
PC-Relative Displacement																							

BRSET #n,[X or Y]:aa,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	0	a	a	a	a	a	a	1	S	1	b	b	b	b	b
PC-Relative Displacement																							

BRSET #n,[X or Y]:pp,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	1	p	p	p	p	p	p	0	S	1	b	b	b	b	b
PC-Relative Displacement																							

BRSET #n,[X or Y]:qq,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	0	q	q	q	q	q	q	0	S	1	b	b	b	b	b
PC-Relative Displacement																							

BRSET #n,S,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	1	D	D	D	D	D	D	1	0	1	b	b	b	b	b
PC-Relative Displacement																							

BScc Branch to Subroutine Conditionally BScc

Operation

If	cc,	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1fiPC
If	cc,	then	PC → SSH;SR → SSL;PC + xxx → PC
		else	PC + 1 → PC
If	cc,	then	PC → SSH;SR → SSL;PC + Rn → PC
		else	PC + 1 → PC

Assembler Syntax

BScc xxxx

BScc xxx

BScc Rn

Instruction Fields

{cc}	**CCCC**	Condition code (see **Table 12-18** on page 12-28)
{xxxx}		24-bit PC-Relative Long Displacement
{xxx}	**aaaaaaaaa**	Signed PC-Relative Short Displacement
{Rn}	**RRR**	Address register [R0 – R7]

Description If the specified condition is true, the address of the instruction immediately following the BScc instruction and the SR are pushed onto the stack. Program execution then continues at location PC + displacement. If the specified condition is false, the PC is incremented and program execution continues sequentially. The displacement is a 2's complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Short Displacement and Address Register PC Relative addressing modes may be used. The Short Displacement 9-bit data is sign extended to form the PC relative displacement. The conditions that the term "cc" can specify are listed on **Table 12-18** on page 12-28.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

		23 16	15 8	7 0
BScc	xxxx	0 0 0 0 1 1 0 1	0 0 0 1 0 0 0 0	0 0 0 0 C C C C
		PC-Relative Displacement		

		23 16	15 8	7 0
BScc	xxx	0 0 0 0 0 1 0 1	C C C C 0 0 a a	a a 0 a a a a a

		23 16	15 8	7 0
BScc	Rn	0 0 0 0 1 1 0 1	0 0 0 1 1 R R R	0 0 0 0 C C C C

BSCLR Branch to Subroutine if Bit Clear BSCLR

Operation

If	S{n}=0	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1fiPC
If	S{n}=0	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1fiPC
If	S{n}=0	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1fiPC
If	S{n}=0	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1fiPC
If	S{n}=0	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1fiPC

Assembler Syntax

BSCLR	#n,[X or Y]:ea,xxxx
BSCLR	#n,[X or Y],aa,xxxx
BSCLR	#n,[X or Y]:pp,xxxx
BSCLR	#n,[X or Y]:qq,xxxx
BSCLR	#n,S,xxxx

Instruction Fields

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address
{X/Y}	**S**	Memory Space [X,Y]
{xxxx}		24-bit Relative Long Displacement
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers]

See **Table 12-13** on page 12-22

Description The n^{th} bit in the source operand is tested. If the tested bit is cleared, the address of the instruction immediately following the BSCLR instruction and the status register are pushed onto the stack. Program execution then continues at location PC+displacement. If the tested bit is set, the PC is incremented and program execution continues sequentially. However, the address register specified in the effective address field is always updated independently of the condition. The displacement is a two's complement 24-bit integer that represents the relative distance from the current PC to the destination PC. The 24-bit displacement is contained in the extension word of the instruction. All memory alterable addressing modes can reference the source operand. Absolute Short, I/O Short and Register Direct addressing modes can also be used. Note that if the specified source operand S is the SSH, the stack pointer register decrements by

one; if the condition is true, the push operation writes over the stack level where the SSH value is taken. The bit to be tested is selected by an immediate bit number 0-23.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition

— Unchanged by the instruction

Instruction Formats and opcodes

BSCLR #n,[X or Y]:ea,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	0	M	M	M	R	R	R	0	S	0	b	b	b	b	b
PC-Relative Displacement																							

BSCLR #n,[X or Y]:aa,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	0	a	a	a	a	a	a	1	S	0	b	b	b	b	b
PC-Relative Displacement																							

BSCLR #n,[X or Y]:qq,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	0	q	q	q	q	q	q	1	S	0	b	b	b	b	b
PC-Relative Displacement																							

BSCLR #n,[X or Y]:pp,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	1	p	p	p	p	p	p	0	S	0	b	b	b	b	b
PC-Relative Displacement																							

BSCLR #n,S,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	1	D	D	D	D	D	D	1	0	0	b	b	b	b	b
PC-Relative Displacement																							

BSET — Bit Set and Test — BSET

Operation

D[n] → C	1→ D[n]
D[n] → C	1 → D[n]
D[n] → C	1 → D[n]
D[n] → C	1 → D[n]
D[n] → C	1 → D[n]

Assembler Syntax

BSET	#n,[XorY]:ea
BSET	#n,[XorY]:aa
BSET	#n,[XorY]:pp
BSET	#n,[XorY]:qq
BSET	#n,D

Instruction Fields

{#n}	**bbbb**	Bit number [0–23]	See **Table 12-13** on page 12-22
{ea}	**MMMRRR**	Effective Address	
{X/Y}	**S**	Memory Space [X,Y]	
{aa}	**aaaaaa**	Absolute Address [0–63]	
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF]	
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]	
{D}	**DDDDDD**	Destination register [all on-chip registers]	

Description Test the n^{th} bit of the destination operand D, set it, and store the result in the destination location. The state of the n^{th} bit is stored in the Carry bit (C) of the CCR register. The bit to be tested is selected by an immediate bit number from 0–23. This instruction performs a read-modify-write operation on the destination location using two destination accesses before releasing the bus. This instruction provides a test-and-set capability that is useful for synchronizing multiple processors using a shared memory. This instruction can use all memory alterable addressing modes. When this instruction performs a bit manipulation/test on either the A or B 56-bit accumulator, it optionally shifts the accumulator value according to scaling mode bits S0 and S1 in the system Status Register (SR). If the data out of the shifter indicates that the accumulator extension register is in use, the instruction acts on the limited value (limited on the maximum positive or negative saturation constant). The “L” flag in the SR is set accordingly.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

CCR Condition Codes

For destination operand SR:

* C Set if bit 0 is specified, unaffected otherwise.
* V Set if bit 1 is specified, unaffected otherwise.
* Z Set if bit 2 is specified, unaffected otherwise.
* N Set if bit 3 is specified, unaffected otherwise.
* U Set if bit 4 is specified, unaffected otherwise.
* E Set if bit 5 is specified, unaffected otherwise.
* L Set if bit 6 is specified, unaffected otherwise.
* S Set if bit 7 is specified, unaffected otherwise.

For other destination operands:

* C Set if bit tested is set, and cleared otherwise.
* V Unaffected.
* Z Unaffected.
* N Unaffected.
* U Unaffected.
* E Unaffected.
* L Set according to the standard definition.
* S Set according to the standard definition.

MR Status Bits

For destination operand SR:

* I0 Changed if bit 8 is specified, unaffected otherwise.
* I1 Changed if bit 9 is specified, unaffected otherwise.
* S0 Changed if bit 10 is specified, unaffected otherwise.
* S1 Changed if bit 11 is specified, unaffected otherwise.
* FV Changed if bit 12 is specified, unaffected otherwise.
* SM Changed if bit 13 is specified, unaffected otherwise.
* RM Changed if bit 14 is specified, unaffected otherwise.
* LF Changed if bit 15 is specified, unaffected otherwise.

For other destination operands: MR status bits are not affected.

Instruction Formats and opcodes

BSET #n,[X or Y]:ea

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	M	M	M	R	R	R	0	S	1	0	b	b	b	b
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

BSET #n,[X or Y]:aa

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	a	a	a	a	a	a	0	S	1	0	b	b	b	b

BSET #n,[X or Y]:pp

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	0	p	p	p	p	p	p	0	S	1	0	b	b	b	b

BSET #n,[X or Y]:qq

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	q	q	q	q	q	q	0	S	1	0	b	b	b	b

BSET #n,D

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	1	D	D	D	D	D	D	0	1	1	0	b	b	b	b

BSR — Branch to Subroutine — BSR

Operation

PC fiSSH;SR fiSSL;PC+xxxxfiPC

PC → SSH;SR →SSL;PC + xxx → PC

PC → SSH;SR → SSL;PC + Rn → PC

Assembler Syntax

BSR xxxx

BSR xxx

BSR Rn

Instruction Fields

{xxxx}		24-bit PC-Relative Long Displacement
{xxx}	**aaaaaaaaa**	Signed PC-Relative Short Displacement
{Rn}	**RRR**	Address register [R0–R7]

Description The address of the instruction immediately following the BSR instruction and the SR are pushed onto the stack. Program execution then continues at location PC + displacement. The displacement is a twos-complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Short Displacement and Address Register PC-Relative addressing modes can be used. The Short Displacement 9-bit data is sign-extended to form the PC-Relative displacement.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

BSR xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
PC-Relative Displacement																							

BSR xxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0	0	0	0	1	0	a	a	a	a	0	a	a	a	a	a

BSR Rn

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0	0	0	1	1	R	R	R	1	0	0	0	0	0	0	0

BSSET Branch to Subroutine if Bit Set BSSET

Operation

If	S{n}=1	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1⇒PC
If	S{n}=1	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1⇒PC
If	S{n}=1	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1⇒PC
If	S{n}=1	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1⇒PC
If	S{n}=1	then	PC fiSSH;SR fiSSL;PC+xxxx fiPC
		else	PC+1⇒PC

Assembler Syntax

BSSET	#n,[X or Y]:ea,xxxx
BSSET	#n,[X or Y],aa,xxxx
BSSET	#n,[X or Y]:pp,xxxx
BSSET	#n,[X or Y]:qq,xxxx
BSSET	#n,S,xxxx

Instruction Fields

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address
{X/Y}	**S**	Memory Space [X,Y]
{xxxx}		24-bit Relative Long Displacement
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers]

See **Table 12-13** on page 12-22

Description The n^{th} bit in the source operand is tested. If the tested bit is set, the address of the instruction immediately following the BSSET instruction and the status register is pushed onto the stack. Program execution then continues at location PC+displacement. If the tested bit is cleared, the PC is incremented and program execution continues sequentially. However, the address register specified in the effective address field is always updated independently of the condition. The displacement is a two's complement 24-bit integer that represents the relative distance from the current PC to the destination PC. The 24-bit displacement is contained in the extension word of the instruction. All memory alterable addressing modes can reference the source operand. Absolute Short, I/O Short and Register Direct addressing modes can also be used. Note that if the specified source operand S is the SSH, the stack pointer register is decremented by one; if the condition is true, the push operation writes over the stack level where the SSH value is taken. The bit to be tested is selected by an immediate bit number 0-23.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

BSSET #n,[X or Y]:ea,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	0	M	M	M	R	R	R	0	S	1	b	b	b	b	b
PC-Relative Displacement																							

BSSET #n,[X or Y]:aa,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	0	a	a	a	a	a	a	1	S	1	b	b	b	b	b
PC-Relative Displacement																							

BSSET #n,[X or Y]:pp,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	1	p	p	p	p	p	p	0	S	1	b	b	b	b	b
PC-Relative Displacement																							

BSSET #n,[X or Y]:qq,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	0	q	q	q	q	q	q	1	S	1	b	b	b	b	b
PC-Relative Displacement																							

BSSET #n,S,xxxx

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	1	D	D	D	D	D	D	1	0	1	b	b	b	b	b
PC-Relative Displacement																							

BTST — Bit Test — BTST

Operation

D[n] → C	
D[n] → C	
D[n] → C	
D[n] → C	
D[n] → C	

Assembler Syntax

BTST	#n,[XorY]:ea
BTST	#n,[XorY]:aa
BTST	#n,[XorY]:pp
BTST	#n,[XorY]:qq
BTST	#n,D

Instruction Fields

{#n}	**bbbb**	Bit number [0 – 23]
{ea}	**MMMRRR**	Effective Address
{X/Y}	**S**	Memory Space [X,Y]
{aa}	**aaaaaa**	Absolute Address [0–63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0 – $FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80 – $FFFFBF]
{D}	**DDDDDD**	Destination register [all on-chip registers]

See **Table 12-13** on page 12-22

Description Test the nth bit of the destination operand D. The state of the nth bit is stored in the Carry bit (C) of the CCR. The bit to test is selected by an immediate bit number from 0–23. BTST is useful for performing serial-to-parallel conversion with appropriate rotate instructions. This instruction can use all memory alterable addressing modes.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	—	—	—	*
CCR							

* C Set if bit tested is set, and cleared otherwise.
Changed according to the standard definition.

— Unchanged by the instruction.

SP—Stack Pointer

For destination operand SSH:SP, decrement the SP by 1.

For other destination operands, the SPis not affected.

Instruction Formats and opcodes

BTST #n,[X or Y]:ea

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	1	M	M	M	R	R	R	O	S	1	0	b	b	b	b
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

BTST #n,[X or Y]:aa

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	0	a	a	a	a	a	a	0	S	1	0	b	b	b	b

BTST #n,[X or Y]:pp

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	0	p	p	p	p	p	p	0	S	1	0	b	b	b	b

BTST #n,[X or Y]:qq

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	q	q	q	q	q	q	0	S	1	0	b	b	b	b

BTST #n,D

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	1	D	D	D	D	D	D	0	1	1	0	b	b	b	b

CLB — Count Leading Bits — CLB

Operation

If S[39] = 0 then
9 – (Number of consecutive leading zeros in S[55:0]) → D[47:24]
else
9 – (Number of consecutive leading ones in S[55:0]) → D[47:24]

Assembler Syntax

CLB S,D

Instruction Fields

{D}	D	Destination accumulator [A,B]
{S}	S	Source accumulator [A,B]

See **Table 12-13** on page 12-22

Description Count leading 0s or 1s according to Bit 55 of the source accumulator. Scan bits 55–0 of the source accumulator starting from Bit 55. The MSP of the destination accumulator is loaded with nine minus the number of consecutive leading 1s or 0s found. The result is a signed integer in MSP whose range of possible values is from +8 to –47. This is a 56-bit operation. The LSP of the destination accumulator D is filled with 0s. The EXP of the destination accumulator D is sign-extended.

Note:

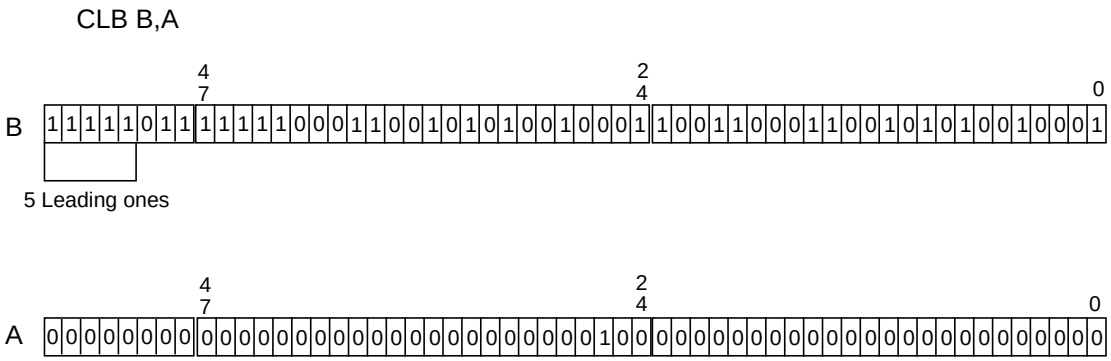
1. If the source accumulator is all 0s, the result is 0.
2. In Sixteen-Bit Arithmetic mode, the count ignores the unused 8 Least Significant Bits of the MSP and LSP of the source accumulator. Therefore, the result is a signed integer whose range of possible values is from +8 to –31.
3. CLB can be used in conjunction with NORMF instruction to specify the shift direction and amount needed for normalization.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	*	*	*	—
CCR							

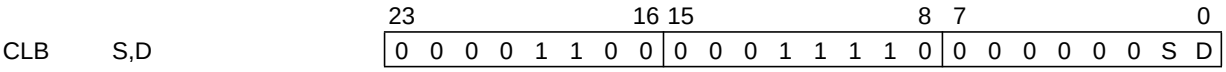
*	N	Set if bit 47 of the result is set, and cleared otherwise.
*	Z	Set if bits 47–24 of the result are all 0.
*	V	Always cleared.
—		Unchanged by the instruction.

Example



Result in A is 9 - 5 = 4

Instruction Formats and opcode



CLR — Clear Accumulator — CLR

Operation

0 → D (parallel move)

Assembler Syntax

CLR D (parallel move)

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description

Clear the destination accumulator. This is a 56-bit clear instruction.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	*	*	*	*	*	—
CCR							

* E Always cleared.
* U Always set.
* N Always cleared.
* Z Always set.
* V Always cleared.
* √ Changed according to the standard definition.
— Unchanged by the instruction.

Instruction Formats and opcodes

CLR D

23 — 16	15 — 8	7 — 0
Data Bus Move Field	0 0 0 1	d 0 1 1
Optional Effective Address Extension		

CMP Compare CMP

Operation

S2 – S1 (parallel move)

S2 – #xx

S2 – #xxxxxx

Assembler Syntax

CMP S1, S2 (parallel move)

CMP #xx, S2

CMP #xxxxxx, S2

Instruction Fields

{S1}	**JJJ**	Source one register [B/A,X0,Y0,X1,Y1] (see **Table 12-16** on page 12-24)
{S2}	**d**	Source two accumulator [A/B] (see **Table 12-13** on page 12-22)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxxxx}		24-bit Immediate Long Data extension word

Description Subtract the source one operand from the source two accumulator, S2, and update the CCR. The result of the subtraction operation is not stored. The source one operand can be a register (24-bit word or 56-bit accumulator), 6-bit short immediate, or 24-bit long immediate. When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the six bits will be right-aligned and the remaining bits will be zeroed to form a 24-bit source operand.

This instruction subtracts 56-bit operands. When a word is specified as the source one operand, it is sign-extended and zero-filled to form a valid 56-bit operand. For the carry to be set correctly as a result of the subtraction, S2 must be properly sign-extended. S2 can be improperly sign-extended by writing A1 or B1 explicitly prior to executing the compare so that A2 or B2, respectively, may not represent the correct sign extension. This particularly applies to the case where it is extended to compare 24-bit operands, such as X0 with A1.

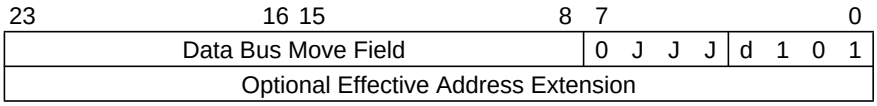
Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	√
CCR							

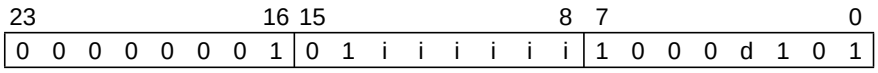
√ Changed according to the standard definition.

Instruction Formats and opcodes

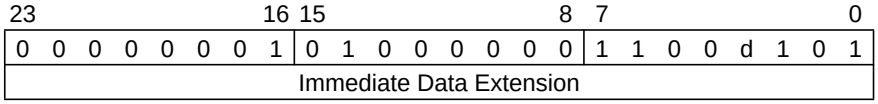
CMP S1, S2



CMP #xx, S2



CMP #xxxx,S2



CMPM Compare Magnitude CMPM

Operation

|S2| – |S1| (parallel move)

Assembler Syntax

CMPM S1, S2 (parallel move)

Instruction Fields

{S1} **JJJ** Source one register [B/A,X0,Y0,X1,Y1] (see **Table 12-16** on page 12-24)

{S2} **d** Source two accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Subtract the absolute value (magnitude) of the source one operand, S1, from the absolute value of the source two accumulator, S2, and update the CCR. The result of the subtraction operation is not stored. Note that this instruction subtracts 56-bit operands. When a word is specified as S1, it is sign-extended and zero-filled to form a valid 56-bit operand. For the carry to be set correctly as a result of the subtraction, S2 must be properly sign-extended. S2 can be improperly sign-extended by writing A1 or B1 explicitly prior to executing the compare so that A2 or B2, respectively, may not represent the correct sign extension. This applies especially when it is extended to compare 24-bit operands, such as X0 with A1.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	√	√	√	√	√	√
CCR							

√ Changed according to the standard definition.

Instruction Formats and opcodes

CMPM S1, S2

23 16	15 8	7							0
Data Bus Move Field		0	J	J	J	d	1	1	1
Optional Effective Address Extension									

CMPU Compare Unsigned CMPU

Operation

S2 – S1

Assembler Syntax

CMPU S1, S2

Instruction Fields

{S1}	**ggg**	Source one register [A,B,X0,Y0,X1,Y1]	See **Table 12-13** on page 12-22
{S2}	**d**	Source two accumulator [A,B]	

Description Subtract the source one operand, S1, from the source two accumulator, S2, and update the CCR. The result of the subtraction operation is not stored. Note that this instruction subtracts a 24- or 48-bit unsigned operand from a 48-bit unsigned operand. When a 24-bit word is specified as S1, it is aligned to the left and zero-filled to form a valid 48-bit operand. If an accumulator is specified as an operand, the value in the EXP does not affect the operation.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	√	*	*	√
CCR							

* V Always cleared.

* Z Set if bits 47–0 of the result are 0.

— Unchanged by the instruction.

√ Changed according to the standard definition.

Instruction Formats and opcodes

	23							16	15							8	7							0
CMPU S1, S2	0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	g	g	g	d

DEBUG Enter Debug Mode DEBUG

Operation

Enter the Debug mode

Assembler Syntax

DEBUG

Instruction Fields None

Description Enter the Debug mode and wait for OnCE commands.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

	23								16	15							8	7							0
DEBUG	0	0	0	0	0	0	0	0		0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Enter Debug Mode Conditionally

Operation

If cc, then enter the Debug mode

Assembler Syntax

DEBUGcc

Instruction Fields

{cc} **CCCC** Condition code (see **Table 12-18** on page 12-28)

Description If the specified condition is true, enter the Debug mode and wait for OnCE commands. If the specified condition is false, continue with the next instruction. The conditions that the term “cc” can specify are listed on **Table 12-18** on page 12-28.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

	23							16	15							8	7							0
DEBUGcc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	C	C	C	C

DEC — Decrement by One — DEC

Operation

D – 1 → D

Assembler Syntax

DEC D

Instruction Fields

{D} **d** Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Decrement by one the specified operand and store the result in the destination accumulator. One is subtracted from the LSB of D.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	√	√
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

	23							16	15							8	7							0
DEC D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	d

DIV — Divide Iteration — DIV

Operation

IF D[39]⊕S[15] = 1

then 2 ∗ D + C + S → D

else 2 ∗ D + C – S → D

where ⊕ denotes the logical exclusive OR operator.

Assembler Syntax

DIV S,D

Instruction Fields

{S}	**JJ**	Source input register [X0,X1,Y0,Y1]	See **Table 12-13** on page 12-22
{D}	**d**	Destination accumulator [A,B]	

Description Divide the destination operand D by the source operand S and store the result in the destination accumulator D. The 48-bit dividend must be a positive fraction that is sign-extended to 56 bits and stored in the full 56-bit destination accumulator D. The 24-bit divisor is a signed fraction stored in the source operand S. Each DIV iteration calculates one quotient bit using a nonrestoring fractional division algorithm. After the first DIV instruction executes, the destination operand holds both the partial remainder and the formed quotient. The partial remainder occupies the high-order portion of the destination accumulator D and is a signed fraction. The formed quotient occupies the low-order portion of the destination accumulator D (A0 or B0) and is a positive fraction. One bit of the formed quotient is shifted into the LSB of the destination accumulator at the start of each DIV iteration. The formed quotient is the true quotient if the true quotient is positive. If the true quotient is negative, the formed quotient must be negated. Valid results are obtained only when |D| < |S| and the operands are interpreted as fractions. This condition ensures that the magnitude of the quotient is less than 1 (i.e., a fractional quotient) and precludes division by 0.

DIV calculates one quotient bit based on the divisor and the previous partial remainder. To produce an N-bit quotient, the DIV instruction executes N times, where N is the number of bits of precision desired in the quotient, 1 ≤ N ≤ 24. Thus, for a full-precision (24-bit) quotient, sixteen DIV iterations are required. In general, executing the DIV instruction N times produces an N-bit quotient and a 48-bit remainder that has (48 – N) bits of precision and whose N MSBs are 0s. The partial remainder is not a true remainder and must be corrected due to the nonrestoring nature of the division algorithm before it can be used. Therefore, once the divide is complete, it is necessary to reverse the last DIV operation and restore the remainder to obtain the true remainder.

DIV uses a nonrestoring fractional division algorithm that consists of the following operations:

1. **Compare the source and destination operand sign bits:** An exclusive OR operation is performed on Bit 55 of the destination operand D and Bit 23 of the source operand S.
2. **Shift the partial remainder and the quotient:** The 39-bit destination accumulator D is shifted one bit to the left. The Carry bit (C) is moved into the LSB (Bit 0) of the accumulator.
3. **Calculate the next quotient bit and the new partial remainder:** The 24-bit source operand S (signed divisor) is either added to or subtracted from the Most Significant Portion (MSP) of the destination accumulator (A1 or B1), and the result is stored back into the MSP of that destination accumulator. If the result of the exclusive OR operation previously described was 1 (i.e., the sign bits were different), the source operand S is added to the accumulator. If the result of the exclusive OR operation was 0 (i.e., the sign bits were the same), the source operand S is subtracted from the accumulator. Because of the automatic sign extension of the 24-bit signed divisor, the addition or subtraction operation correctly sets the C bit with the next quotient bit.

For extended precision division (e.g., N-bit quotients where N > 24), the DIV instruction is no longer applicable, and a user-defined N-bit division routine is required. For more information on division algorithms, see pages 524–530 of *Theory and Application of Digital Signal Processing* by Rabiner and Gold (Prentice-Hall, 1975), pages 190–199 of *Computer Architecture and Organization* by John Hayes (McGraw-Hill, 1978), pages 213–223 of *Computer Arithmetic: Principles, Architecture, and Design* by Kai Hwang (John Wiley and Sons, 1979), or other references as required.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	*	—	—	—	—	*	*
CCR							

* L Set if the Overflow bit (V) is set.

* V Set if the MSB of the destination operand is changed as a result of the instruction's left shift operation.

* C Set if Bit 55 of the result is cleared.

— Unchanged by the instruction

Instruction Formats and opcodes

	23							16	15							8	7							0
DIV S,D	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	J	J	d	0	0	0

DMAC DMAC

Double-Precision Multiply-Accumulate With Right Shift

Operation

[D >> 16] ±S1 * S2 → D
(S1 signed, S2 signed)

[D >> 16] ±S1 * S2 → D
(S1 signed, S2 unsigned)

[D >> 16] ±S1 * S2 → D
(S1 unsigned, S2 unsigned)

Assembler Syntax

DMACss	(±)S1,S2,D	(no parallel move)
DMACsu	(±)S1,S2,D	(no parallel move)
DMACuu	(±)S1,S2,D	(no parallel move)

Instruction Fields

{S1,S2}	**QQQQ**	Source registers S1,S2 [all combinations of X0,X1,Y0, and Y1] (see **Table 12-16** on page 12-24)
{D}	**d**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{±}	**k**	Sign [+,–] (see **Table 12-16** on page 12-24)
{ss,su,uu}	**ss**	[ss,su,uu] (see **Table 12-16** on page 12-24)

Description Multiply the two 24-bit source operands S1 and S2 and add/subtract the product to/from the specified 56-bit destination accumulator D, which has been previously shifted 24 bits to the right. The multiplication can be performed on signed numbers (ss), unsigned numbers (uu), or mixed (unsigned * signed, (su)). The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+". This instruction is optimized for multi-precision multiplication support.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	√	—
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

DMAC (±)S1,S2,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	s	1	s	d	k	Q	Q	Q	Q

DO Start Hardware Loop DO

Operation

SP + 1 → SP;LA → SSH;LC → SSL;[X or Y]:ea → LC
SP + 1 → SP;PC → SSH;SR → SSL;expr – 1 → LA
1 → LF

SP + 1 → SP;LA → SSH;LC → SSL;[X or Y]:aa → LC
SP+1 → SP;PC → SSH;SR → SSL;expr – 1 → LA
1 → LF

SP + 1 → SP;LA → SSH;LC → SSL;#xxx → LC
SP+1 → SP;PC → SSH;SR → SSL;expr – 1 → LA
1 → LF

SP + 1 → SP;LA → SSH;LC → SSL;S → LC
SP + 1 → SP;PC → SSH;SR → SSL;expr – 1 → LA
1 → LF

End of Loop:
SSL(LF) → SR;SP – 1 → SP
SSH → LA;SSL → LC;SP – 1 → SP

Assembler Syntax

DO [X or Y]:ea,expr

DO [X or Y]:aa,expr

DO #xxx,expr

DO S,expr

Instruction Fields

{ea}	**MMMRRR**	Effective Address
{X/Y}	**S**	Memory Space [X,Y]
{expr}		24-bit Absolute Address in 16-bit extension word
{aa}	**aaaaaa**	Absolute Address [0–63]
{#xxx}	**hhhhiiiiiiii**	Immediate Short Data [0–4095]
{S}	**DDDDDD**	Source register [all on-chip registers, **except SSH**]

See **Table 12-13** on page 12-22

For the DO SP, expr instruction, the actual value that is loaded into the Loop Counter (LC) is the value of the Stack Pointer (SP) before the DO instruction executes, incremented by 1. Thus, if SP = 3, the execution of the DO SP,expr instruction loads the LC with the value LC = 4. For the DO SSL, expr instruction, the LC is loaded with its previous value, which was saved on the stack by the DO instruction itself.

Description Begin a hardware DO loop that is to be repeated the number of times specified in the instruction's source operand and whose range of execution is terminated by the destination operand (previously shown as "expr"). No overhead other than the execution of this DO instruction is required to set up this loop. DO loops can be nested and the loop count can be passed as a parameter.

During the first instruction cycle, the current contents of the Loop Address (LA) and the Loop Counter (LC) registers are pushed onto the System Stack. The DO source operand then loads into the LC register, which contains the remaining number of times the DO loop is to execute and can be accessed from inside the DO loop under certain restrictions. If the initial value of LC is 0 and the Sixteen-Bit Compatibility mode bit (bit 13, SC, in the Chip Status Register) is cleared, the DO loop does not execute.If LC initial value is zero but SC is set, the DO loop executes 65,536 times. All address register indirect addressing modes can be used to generate the effective address of the source operand. If immediate short data is specified, the twelve LSBs of the LC register are loaded with the 12-bit immediate value, and the twelve MSBs of the LC register are cleared.

During the second instruction cycle, the current contents of the Program Counter (PC) register and the Status Register (SR) are pushed onto the System Stack. The stacking of the LA, LC, PC, and SR registers is the mechanism that permits the nesting of DO loops. The DO destination operand (shown as “expr”) is then loaded into the LA register. This 24-bit operand is located in the instruction’s 24-bit absolute address extension word, as shown in the opcode section. The value in the PC register pushed onto the system stack is the address of the first instruction following the DO instruction (i.e., the first actual instruction in the DO loop). This value is read (copied but not pulled) from the top of the system stack to return to the top of the loop for another pass through the loop.

During the third instruction cycle, the Loop Flag (LF) is set, resulting in a repeated comparison of PC with LA to determine whether the last instruction in the loop has been fetched. If LA equals PC, the last instruction in the loop has been fetched and the LC is tested. If the LC is not equal to 1, it is decremented by one and SSH is loaded into the PC to fetch the first instruction in the loop again. When LC = 1, the “end-of-loop” processing begins.

When a DO loop executes , the instructions are actually fetched each time through the loop. Therefore, a DO loop can be interrupted. DO loops can also be nested. When DO loops are nested, the end-of-loop addresses must also be nested and are not allowed to be equal. The assembler generates an error message when DO loops are improperly nested.

During the “end-of-loop” processing, the Loop Flag (LF) from the lower portion (SSL) of the Stack Pointer is written into the SR, the contents of the LA register are restored from the upper portion (SSH) of (SP – 1), the contents of LC are restored from the lower portion (SSL) of (SP – 1), and the Stack Pointer is decremented by two. Instruction fetches continue at the address of the instruction following the last instruction in the DO loop. Note that LF is the only bit in the SR that is restored after a hardware DO loop is exited.

Note:

1. The assembler calculates the end-of-loop address to be loaded into LA (the absolute address extension word) by evaluating the end-of-loop expression “expr” and subtracting 1. This is done to accommodate the case where the last word in the DO loop is a two-word instruction. Thus, the end-of-loop expression “expr” in the source code must represent the address of the instruction AFTER the last instruction in the loop.
2. The Loop Flag (LF) is cleared by a hardware reset.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	—	—	—	—	—	—
CCR							

* S Set if the instruction sends A/B accumulator contents to XDB or YDB.

* L Set if data limiting occurred [see Note].

— Unchanged by the instruction.

Instruction Formats and opcodes

DO [X or Y]:ea, expr

23								16	15							8	7							0
0	0	0	0	0	1	1	0		0	1	M	M	M	R	R	R	0	S	0	0	0	0	0	0
Absolute Address Extension Word																								

DO [X or Y]:aa, expr

23								16	15							8	7							0
0	0	0	0	0	1	1	0		0	0	a	a	a	a	a	a	0	S	0	0	0	0	0	0
Absolute Address Extension Word																								

DO #xxx, expr

23								16	15							8	7							0
0	0	0	0	0	1	1	0		i	i	i	i	i	i	i	i	1	0	0	0	h	h	h	h
Absolute Address Extension Word																								

DO S, expr

23								16	15							8	7							0
0	0	0	0	0	1	1	0		1	1	D	D	D	D	D	D	0	0	0	0	0	0	0	0
Absolute Address Extension Word																								

Start Infinite Loop

Operation

SP + 1 → SP;LA → SSH;LC → SSL
SP + 1 → SP;PC → SSH;SR → SSL;expr – 1 → LA
1 → LF; 1 →FV

Assembler Syntax

DO FOREVER,expr

Instruction Fields

None

Description Begin a hardware DO loop that is to repeat forever with a range of execution terminated by the destination operand (“expr”). No overhead other than the execution of this DO FOREVER instruction is required to set up this loop. DO FOREVER loops can nest with other types of instructions. During the first instruction cycle, the contents of the Loop Address (LA) and the Loop Counter (LC) registers are pushed onto the system stack. The LC register is pushed onto the stack but is not updated by this instruction.

During the second instruction cycle, the contents of the Program Counter (PC) register and the Status Register (SR) are pushed onto the system stack. Stacking the LA, LC, PC, and SR registers permits nesting DO FOREVER loops. The DO FOREVER destination operand (shown as “expr”) is then loaded into the LA register. This 24-bit operand resides in the instruction’s 24-bit absolute address extension word, as shown in the opcode section. The value in the PC register pushed onto the system stack is the address of the first instruction following the DO FOREVER instruction (i.e., the first actual instruction in the DO FOREVER loop). This value is read (copied, but not pulled) from the top of the system stack to return to the top of the loop for another pass through the loop.

During the third instruction cycle, the Loop Flag (LF) and the Forever flag are set. Thus, the PC is repeatedly compared with LA to determine whether the last instruction in the loop has been fetched. When LA equals PC, the last instruction in the loop has been fetched and SSH is loaded into the PC to fetch the first instruction in the loop again. The LC register is then decremented by one without being tested. You can use this register to count the number of loops already executed.

Because the instructions are fetched each time through the DO FOREVER loop, the loop can be interrupted. DO FOREVER loops can also be nested. When DO FOREVER loops are nested, the end of loop addresses must also be nested and are not allowed to be equal. The assembler generates an error message when DO FOREVER loops are improperly nested.

DO FOREVER DO FOREVER

Start Infinite Loop

Note:

1. The assembler calculates the end-of-loop address to be loaded into LA (the absolute address extension word) by evaluating the end-of-loop expression “expr” and subtracting one. This is done to accommodate the case where the last word in the DO loop is a two-word instruction. Thus, the end-of-loop expression “expr” in the source code must represent the address of the instruction AFTER the last instruction in the loop.
2. The LC register is never tested by the DO FOREVER instruction, and the only way of terminating the loop process is to use either the ENDDO or BRKcc instructions. LC is decremented every time PC = LA so that it can be used by the programmer to keep track of the number of times the DO FOREVER loop has been executed. If the programer wants to initialize LC to a particular value before the DO FOREVER, care should be taken to save it before if the DO loop is nested. If so, LC should also be restored immediately after exiting the nested DO FOREVER loop.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

DO FOREVER

23								16	15							8	7							0
0	0	0	0	0	0	0	0		0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1
Absolute Address Extension Word																								

DOR Start PC-Relative Hardware Loop DOR

Operation	Assembler Syntax
SP+1 ⇨ SP;LA ⇨ SSH;LC ⇨ SSL;[X or Y]:ea ⇨ LC SP+1 ⇨ SP;PC ⇨ SSH;SR ⇨ SSL;PC+xxxx ⇨ LA 1 ⇨ LF	DOR [Xor Y]:ea,label
SP+1 ⇨ SP;LA ⇨ SSH;LC ⇨ SSL;[X or Y]:ea ⇨ LC SP+1 ⇨ SP;PC ⇨ SSH;SR ⇨ SSL;PC+xxxx ⇨ LA 1 ⇨ LF	DOR [Xor Y]:aa,label
SP+1 ⇨ SP;LA ⇨ SSH;LC ⇨ SSL;#xxx ⇨ LC SP+1 ⇨ SP;PC ⇨ SSH;SR ⇨ SSL;PC+xxxx ⇨ LA 1 ⇨ LF	DOR #xxx,label
SP+1 ⇨ SP;LA ⇨ SSH;LC ⇨ SSL;S ⇨ LC SP+1 ⇨ SP;PC ⇨ SSH;SR ⇨ SSL;PC+xxxx ⇨ LA 1 ⇨ LF	DOR S,label

Instruction Fields

{ea}	**MMMRRR**	Effective Address (see **Table 12-13** on page 12-22)
{X/Y}	**S**	Memory Space [X,Y] (see **Table 12-13** on page 12-22)
{label}		24-bit Address Displacement in 24-bit extension word
{aa}	**aaaaaa**	Absolute Address [0-63]
{#xxx}	**hhhhiiiiiiii**	Immediate Short Data [0-4095]
{S}	**DDDDDD**	Source register [all on-chip registers **except SSH**] (see **Table 12-13** on page 12-22)

Description Initiates the beginning of a PC-relative hardware program loop. The loop address (LA) and loop counter (LC) values are pushed onto the system stack. With proper system stack management, this allows unlimited nested hardware DO loops. The PC and SR are pushed onto the system stack. The PC is added to the 24-bit address displacement extension word and the resulting address is loaded into the loop address register (LA). The effective address specifies the address of the loop count that is loaded into the loop counter (LC). The DO loop executes LC times. If the LC initial value is zero and the 16-Bit Compatibility mode bit (bit 13, SC, in the Status Register) is cleared, the DO loop is not executed. If LC initial value is zero but SC is set, the DO loop executes 65,536 times. All address register indirect addressing modes (less Long Displacement) can be used. Register Direct addressing mode can also be used. If immediate short data is specified, the LC is loaded with the zero extended 12-bit immediate data.

During hardware loop operation, each instruction is fetched each time through the program loop. Therefore, instructions executing in a hardware loop are interruptible and can be nested. The value of the PC pushed onto the system stack is the location of the first

instruction after the DOR instruction. This value is read from the top of the system stack to return to the start of the program loop. When DOR instructions are nested, the end of loop addresses must also be nested and are not allowed to be equal.

The assembler calculates the end of loop address LA (PC-relative address extension word xxxx) by evaluating the end of loop expression and subtracting one. Thus, the end of the loop expression in the source code represents the “next address” after the end of the loop. If a simple end of loop address label is used, it should be placed after the last instruction in the loop.

Since the end of loop comparison occurs at fetch time ahead of the end of loop execution, instructions that change program flow or the system stack cannot be used near the end of the loop without some restrictions. Proper hardware loop operation is guaranteed if no instruction starting at address LA-2, LA-1 or LA specifies the program controller registers SR, SP, SSL, LA, LC or (implicitly) PC as a destination register; or specifies SSH as a source or destination register. Also, SSH cannot be specified as a source register in the DOR instruction itself. The assembler generates a warning if the restricted instructions are found within their restricted boundaries.

Implementation Notes

DOR SP,xxxx The actual value to be loaded into the LC is the value of the SP before the DOR instruction incremented by one.

DOR SSL,xxxx The LC is loaded with its previous value saved in the stack by the DOR instruction itself.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	—	—	—	—	—	—
CCR							

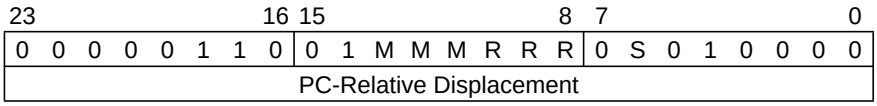
* S Set if the instruction sends A/B accumulator contents to XDB or YDB.

* L Set if data limiting occurred

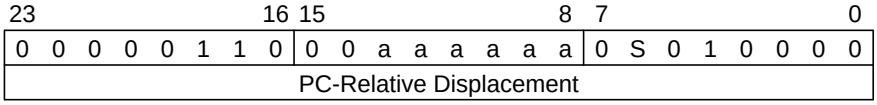
— Unchanged by the instruction

Instruction Formats and opcodes

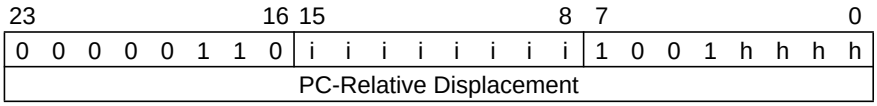
DOR [X or Y]:ea,label



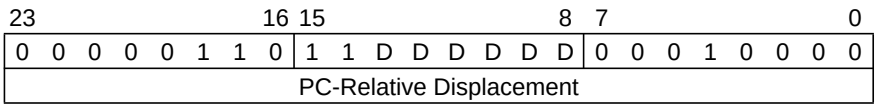
DOR [X or Y]:aa,label



DOR #xxx, label



DOR S, label



DOR FOREVER DOR FOREVER

Start PC-Relative Infinite Loops

Operation

SP+1 fi SP;LA fi SSH;LC fi SSL
SP+1 fi SP;PC fi SSH;SR fi SSL;PC+xxxx fi LA
1 fi LF; 1 fiFV

Assembler Syntax

DOR FOREVER,label

Instruction Fields None.

Description Begin a hardware DO loop that is to repeat forever with a range of execution terminated by the destination operand (“label”). No overhead other than the execution of this DOR FOREVER instruction is required to set up this loop. DOR FOREVER loops can be nested. During the first instruction cycle, the contents of the Loop Address (LA) and the Loop Counter (LC) registers are pushed onto the system stack. The loop counter (LC) register is pushed onto the stack but is not updated.

During the second instruction cycle, the contents of the Program Counter (PC) register and the Status Register (SR) are pushed onto the system stack. Stacking the LA, LC, PC, and SR registers permits nesting DOR FOREVER loops. The DOR FOREVER destination operand (shown as label) is then loaded into the Loop Address (LA) register after it is added to the PC. This 24-bit operand resides in the instruction’s 24-bit relative address extension word as shown in the opcode section. The value in the Program Counter (PC) register pushed onto the system stack is the address of the first instruction following the DOR FOREVER instruction (i.e., the first actual instruction in the DOR FOREVER loop). This value is read (i.e., copied but not pulled) from the top of the system stack to return to the top of the loop for another pass through the loop.

During the third instruction cycle, the Loop Flag (LF) and the ForeVer flag are set. As a result, the PC is repeatedly compared with LA to determine whether the last instruction in the loop has been fetched. If LA equals PC, the last instruction in the loop has been fetched and SSH is read (i.e copied but not pulled) into the PC to fetch the first instruction in the loop again. The loop counter (LC) register is then decremented by one without being tested. You can use this register to count the number of loops already executed.

When a DOR FOREVER loop executes, the instructions are fetched each time through the loop. Therefore, a DOR FOREVER loop can be interrupted. DOR FOREVER loops can also be nested. When DOR FOREVER loops are nested, the end of loop addresses must also be nested and cannot be equal. The assembler generates an error message when DOR FOREVER loops are improperly nested.

DOR FOREVER DOR FOREVER

Start PC-Relative Infinite Loops

Note: The assembler calculates the end of loop address LA (PC-relative address extension word xxxx) by evaluating the end of loop expression and subtracting one. Thus the end of loop expression in the source code represents the “next address” after the end of the loop. If a simple end of loop address label is used, it should be placed after the last instruction in the loop.

The DOR FOREVER instruction never tests the loop counter (LC) register . The only way to terminate the loop process is to use either the ENDDO or BRKcc instruction. LC is decremented every time PC=LA, so you can use it to keep track of the number of times the DOR FOREVER loop has executed. If you want to initialize LC to a particular value before the DOR FOREVER, take care to save it before if the DO loop is nested. If so, LC should also be restored immediately after exiting the nested DOR FOREVER loop.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction

Instruction Formats and opcodes

DOR FOREVER

23								16	15							8	7							0
0	0	0	0	0	0	0	0		0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
PC-Relative Displacement																								

ENDDO End Current DO Loop ENDDO

Operation

SSL(LF) → SR;SP – 1 → SP
SSH → LA; SSL → LC;SP – 1 → SP

Assembler Syntax

ENDDO

Instruction Fields

None

Description Terminate the current hardware DO loop before the current Loop Counter (LC) equals one. If the value of the current DO LC is needed, it must be read before the execution of the ENDDO instruction. Initially, the Loop Flag (LF) is restored from the system stack and the remaining portion of the Status Register (SR) and the Program Counter (PC) are purged from the system stack. The Loop Address (LA) and the LC registers are then restored from the system stack.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

	23							16	15							8	7							0
ENDDO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0

EOR — Logical Exclusive OR — EOR

Operation

S ⊕ D[47:24] → D[47:24] (parallel move)

#xx ⊕ D[47:24] → D[47:24]

#xxxx ⊕ D[47:24] → D[47:24]

where ⊕ denotes the logical XOR operator.

Assembler Syntax

EOR S,D (parallel move)

EOR #xx,D

EOR #xxxx,D

Instruction Fields

{S}	**JJ**	Source register [X0,X1,Y0,Y1]
{D}	**d**	Destination accumulator [A/B]
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxx}		24-bit Immediate Long Data extension word

See **Table 12-13** on page 12-22

Description Logically exclusive OR the source operand S with bits 47:24 of the destination operand D and store the result in bits 47–24 of the destination accumulator. The source can be a 24-bit register, 6-bit short immediate or 24-bit long immediate. This instruction is a 24-bit operation. The remaining bits of the destination operand D are not affected. When 6-bit immediate datais used, the data is interpreted as an unsigned integer. That is, the 6 bits are right-aligned, and the remaining bits are zeroed to form a 24-bit source operand.

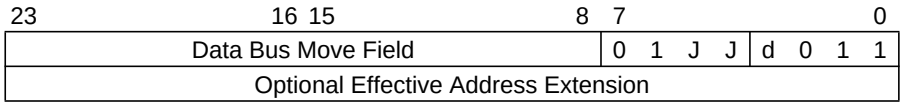
Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
√	√	—	—	*	*	*	—
CCR							

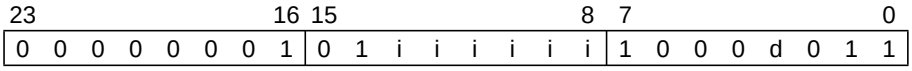
*	N	Set if bit 47 of the result is set.
*	Z	Set if bits 47–24 of the result are 0.
*	V	Always cleared.
√		Changed according to the standard definition.
—		Unchanged by the instruction.

Instruction Formats and opcodes

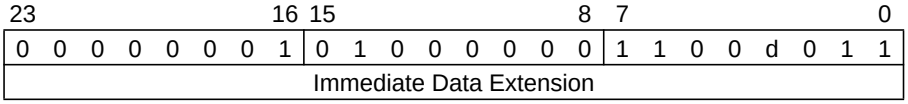
EOR S,D



EOR #xx,D



EOR #xxxx,D



EXTRACT Extract Bit Field EXTRACT

Operation

Offset = S1[5:0]
Width = S1[17:12]

S2[(offset + width – 1):offset] → D[(width – 1):0]
S2[offset + width – 1] → D[39:width] (sign extension)

Offset = #CO[5:0]
Width = #CO[17:12]

S2[(offset + width – 1):offset] → D[(width – 1):0]
S2[offset + width – 1] → D[39:width] (sign extension)

Assembler Syntax

EXTRACT S1,S2,D

EXTRACT #CO,S2,D

Instruction Fields

{S2}	**s**	Source accumulator [A,B]	
{D}	**D**	Destination accumulator [A,B]	See **Table 12-13** on page 12-22
{S1}	**SSS**	Control register [X0,X1,Y0,Y1,A1,B1]	
{#CO}		Control word extension.	

Description Extract a bit-field from source accumulator S2. The bit-field width is specified by bits 17–12 in the S1 register or in the immediate control word #CO. The offset from the Least Significant Bit is specified by bits 5–0 in the S1 register or in the immediate control word #CO. The extracted field is placed into destination accumulator D, aligned to the right. The control register can be constructed by the MERGE instruction. EXTRACT is a 56-bit operation. Bits outside the field are filled with sign extension according to the Most Significant Bit of the extracted bit field.

Note:

1. In Sixteen-bit Arithmetic mode, the offset field is located in bits 13-8 of the control register and the width field is located in bits 21-16 of the control register. These fields corresponds to the definition of the fields in the MERGE instruction.
2. In Sixteen-bit Arithmetic mode, when the width value is zero, then the result will be undefined.
3. If offset + width exceeds the value of 56, the result is undefined.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	√	√	√	√	*	*
CCR							

* V Always cleared.

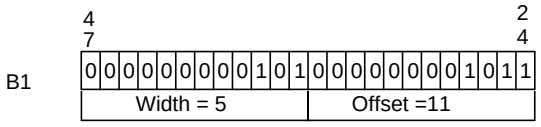
* C Always cleared.

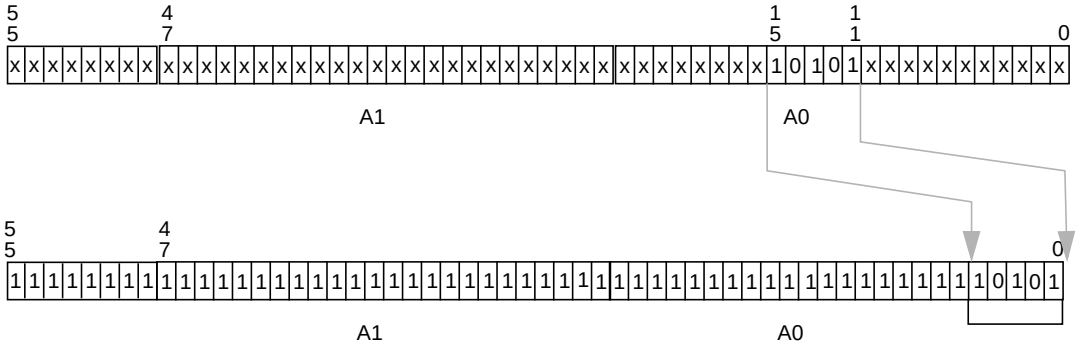
— Unchanged by the instruction.

√ Changed according to the standard definition.

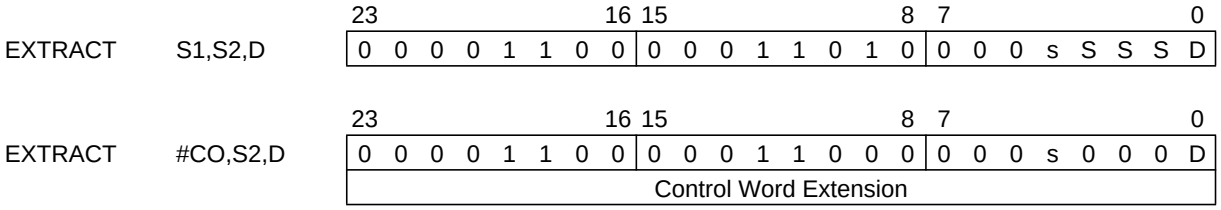
Example

EXTRACT B1,A,A





Instruction Formats and opcodes



EXTRACTU EXTRACTU

Extract Unsigned Bit Field

Operation

Offset = S1[5:0]
Width = S1[17:12]

S2[(offset + width – 1):offset] → D[(width – 1):0]
zero → D[55:width]

Offset = #CO[5:0]
Width = #CO[17:12]

S2[(offset + width – 1):offset] → D[(width – 1):0]
zero fi D[39:width]

Assembler Syntax

EXTRACTU S1,S2,D

EXTRACTU #CO,S2,D

Instruction Fields

{S2}	**s**	Source accumulator [A,B]	
{D}	**D**	Destination accumulator [A,B]	See **Table 12-13** on page 12-22
{S1}	**SSS**	Control register [X0,X1,Y0,Y1,A1,B1]	
{#CO}		Control word extension	

Description Extract an unsigned bit-field from source accumulator S2. The bit-field width is specified by bits 17–12 in the S1 register or in the immediate control word #CO. The offset from the LSB is specified by bits 5–0 in the S1 register or in the immediate control word #CO. The extracted field is placed into destination accumulator D, aligned to the right. The control register can be consructed using the MERGE instruction. EXTRACTU is a 56-bit operation. Bits outside the field are filled with 0s.

Note:

1. In Sixteen-bit Arithmetic mode, the offset field is located in bits 13-8 of the control register and the width field is located in bits 21-16 of the control register. These fields correspond to the definition of the fields in the MERGE instruction.
2. If offset + width exceeds the value of 56, the result is undefined.

Extract Unsigned Bit Field

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	√	√	√	√	*	*
CCR							

* V Always cleared.

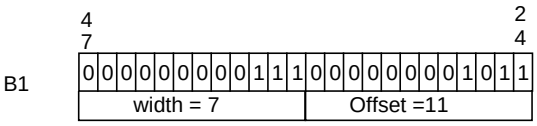
* C Always cleared.

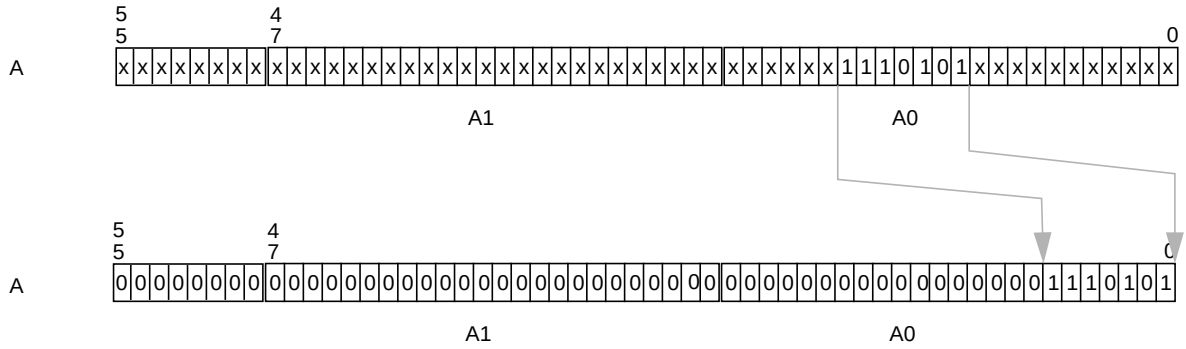
— Unchanged by the instruction.

√ Changed according to the standard definition.

Example

EXTRACTU B1,A,A





Instruction Formats and opcodes

EXTRACTU S1,S2,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	0	1	0	0	s	S	S	S	D

EXTRACTU #CO,S2,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	0	0	s	0	0	0	D
Control Word Extension																							

IFcc Execute Conditionally Without CCR Update IFcc

Operation
If cc, then opcode operation

Assembler Syntax
opcode-Operands IFcc

Instruction Fields

{cc} **CCCC** Condition code (see **Table 12-18** on page 12-28)

Description If the specified condition is true, execute and store result of the specified Data ALU operation. If the specified condition is false, no destination is altered. The CCR is never updated with the condition codes generated by the Data ALU operation. The instructions that can conditionally be executed using IFcc are the parallel arithmetic and logical instructions. See **Table 12-4** on page 12-7 and **Table 12-5** on page 12-9 for a list of those instructions. The conditions specified by “cc” are listed in **Table 12-18** on page 12-28.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

	23							16	15							8	7							0
IFcc	0	0	1	0	0	0	0	0	0	0	1	0	C	C	C	C	Instruction opcode							

IFcc.U Execute Conditionally With CCR Update IFcc.U

Operation
If cc, then opcode operation

Assembler Syntax
opcode-Operands IFcc

Instruction Fields

{cc} **CCCC** Condition code (see **Table 12-18** on page 12-28)

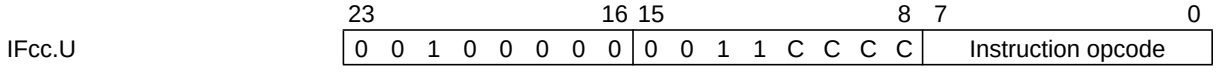
If the specified condition is true, execute and store result of the specified Data ALU operation and update the CCR with the status information generated by the Data ALU operation. If the specified condition is false, no destination is altered and the CCR is not affected. The instructions that can conditionally be executed using IFcc.U are the parallel arithmetic and logical instructions. See **Table 12-4** on page 12-7 and **Table 12-5** on page 12-9 for a list of these instructions. The conditions specified by “cc” are listed on **Table 12-18** on page 12-28

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
*	*	*	*	*	*	*	*
CCR							

* If the specified condition is true, changes are made according to the instruction. Otherwise, it is not changed.

Instruction Formats and opcodes



ILLEGAL Illegal Instruction Interrupt ILLEGAL

Operation
Begin Illegal Instruction exception processing

Assembler Syntax
ILLEGAL

Instruction Fields

None

Description The ILLEGAL instruction executes as if it were a NOP instruction. Normal instruction execution is suspended and illegal instruction exception processing is initiated. The interrupt vector address is located at address P:$3E. The Interrupt Priority Level (I1, I0) is set to 3 in the Status Register if a long interrupt service routine is used. The purpose of the ILLEGAL instruction is to force the DSP into an illegal instruction exception for test purposes. Exiting an illegal instruction is a fatal error. A long exception routine should be used to indicate this condition and cause the system to be restarted.

If the ILLEGAL instruction is in a DO loop at LA and the instruction at LA – 1 is being interrupted, then LC is decremented twice due to the same mechanism that causes LC to be decremented twice if JSR, REP, etc. are located at LA. This is why JSR, REP, and other instructions at LA are restricted. Restrictions cannot be imposed on illegal instructions. Since REP is uninterruptable, repeating an ILLEGAL instruction results in the interrupt not being initiated until after the REP completes. After the interrupt is serviced, program control returns to the address of the second word following the ILLEGAL instruction. Of course, the ILLEGAL interrupt service routine should abort further processing, and the processor should be reinitialized.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	—	—	—	—	—	—	—
CCR							

— Unchanged by the instruction.

Instruction Formats and opcodes

	23								16	15							8	7							0
ILLEGAL	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

INC — Increment by One — INC

Operation

D + 1 → D

Assembler Syntax

INC D

Instruction Fields

{D} d — Destination accumulator [A,B] (see **Table 12-13** on page 12-22)

Description Increment by one the specified operand and store the result in the destination accumulator. One is added from the LSB of D.

Condition Codes

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
—	√	√	√	√	√	√	√
CCR							

√ Changed according to the standard definition.

— Unchanged by the instruction.

Instruction Formats and opcodes

	23							16	15							8	7							0
INC D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	d

INSERT — Insert Bit Field — INSERT

Operation

Offset = S1[5:0]
Width = S1[17:12]

S2[(width – 1):0] → D[(offset + width – 1):offset]

Offset = #CO[5:0]
Width = #CO[17:12]

S2[(width-1):0] → D[(offset + width – 1):offset]

Assembler Syntax

INSERT S1,S2,D

INSERT #CO,S2,D

Instruction Fields

{D}	**D**	Destination accumulator [A,B] (see **Table 12-13** on page 12-22)
{S1}	**SSS**	Control register [X0,X1,Y0,Y1,A1,B1] (see **Table 12-16** on page 12-24)
{S2}	**qqq**	Source register [X0,X1,Y0,Y1,A0,B0] (see **Table 12-16** on page 12-24)
{#CO}		Control word extension

Description Insert a bit-field into the destination accumulator D. The bit-field whose width is specified by bits 17–12 in S1 register begins at the LSB of the S2 register. This bit-field is inserted in the destination accumulator D, with an offset according to bits 5–0 in the S1 register. The S1 operand can be an immediate control word #CO. The width specified by S1 should not exceed a value of 24. The construction of the control register can be done by using the MERGE instruction. This is a 56-bit operation. Any bits outside the field remain unchanged.

Note:

1. In Sixteen-bit Arithmetic mode, the offset field is located in bits 13-8 of the control register and the width field is located in bits 21-16 of the control register. These fields corresponds to the definition of the fields in the MERGE instruction. Width specified by S1 should not exceed a value of 16.
2. In Sixteen-Bit Arithmetic mode, the offset value, located in the offset field, should be the needed offset you pre-incremented by a bias of 16.
3. If offset + width > 56, the result is undefined.