BRA BRA

Branch Always

Operation:

PC+xxxx → Pc

PC+xxx → Pc

PC+Rn → Pc

Assembler Syntax:

BRA xxxx

BRA xxx

BRA Rn

Description:

Program execution continues at location PC+displacement. 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, Long 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.

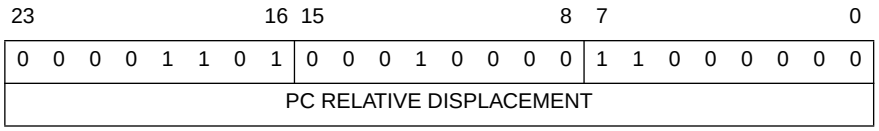
Condition Codes:

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

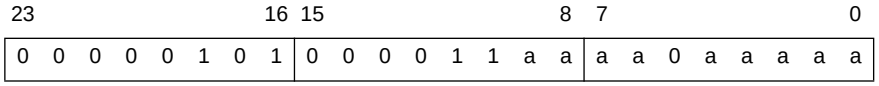
x This bit is unchanged by the instruction

Instruction Formats and opcodes:

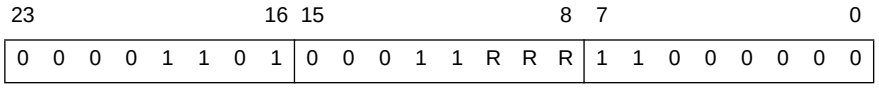
BRA xxxx



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]

BRCLR BRCLR

Branch if bit Clear

Operation:

						Assembler Syntax:	
If	S{n}=0	then	PC+xxxx	→	PC	BRCLR	#n,[X or Y]:ea,xxxx
		else	PC+ 1	→	PC		
If	S{n}=0	then	PC+xxxx	→	PC	BRCLR	#n,[X or Y],aa,xxxx
		else	PC+ 1	→	PC		
If	S{n}=0	then	PC+xxxx	→	PC	BRCLR	#n,[X or Y]:pp,xxxx
		else	PC+ 1	→	PC		
If	S{n}=0	then	PC+xxxx	→	PC	BRCLR	#n,[X or Y]:qq,xxxx
		else	PC+ 1	→	PC		
If	S{n}=0	then	PC+xxxx	→	PC	BRCLR	#n,S,xxxx
		else	PC+ 1	→	PC		

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							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

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

23								16	15							8	7							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								16	15							8	7							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								16	15							8	7							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								16	15							8	7							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								16	15							8	7							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																								

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{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 A-22 on page A-243)

BRKcc BRKcc

Exit Current Do Loop Conditionally

Operation:		**Assembler Syntax:**
If cc	LA+1→PC; SSL(LF,FV)→ SR; SP-1→ SP SSH → LA; SSL→ LC; SP-1→ SP	BRKcc
else	PC+1→ PC	

Description: Exit conditionally the current hardware DO loop before the current loop counter (LC) equals one. It also terminates the DO FOREVER (or DOR FOREVER) loop. If the value of the current DO loop counter (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 loop address (LA) and the loop counter (LC) registers are then restored from the system stack.

The conditions that the term "**cc**" can specify are listed on Table A-43 on page A-251.

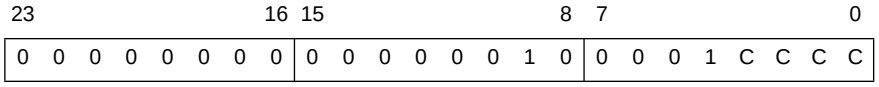
Condition Codes:

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

x This bit is unchanged by the instruction

Instruction Formats and opcodes:

BRKcc



Instruction Fields:

{cc} **CCCC** Condition code (see Table A-43 on page A-251)

BRSET BRSET

Branch if bit Set

Operation:

						Assembler Syntax:	
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,[X or Y]:ea,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,[X or Y],aa,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,[X or Y]:pp,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,[X or Y]:qq,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,S,xxxx
		else	PC+ 1	→	PC		

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							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

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

23								16	15							8	7							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								16	15							8	7							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								16	15							8	7							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								16	15							8	7							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								16	15							8	7							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																								

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{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 A-22 on page A-243)

BScc BScc

Branch to Subroutine Conditionally

Operation:	Assembler Syntax:
If cc, then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BScc xxxx
If cc, then PC →SSH;SR →SSL;PC+xxx →PC else PC+1→PC	BScc xxx
If cc, then PC →SSH;SR →SSL;PC+Rn →PC else PC+1→PC	BScc Rn

Description: If the specified condition is true, the address of the instruction immediately following the BScc instruction and the status register 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, Long 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 A-42 on page A-250.

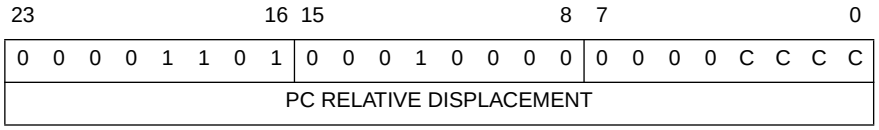
Condition Codes:

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

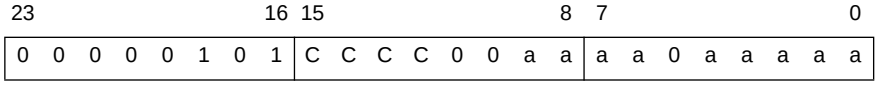
x This bit is unchanged by the instruction

Instruction Formats and opcodes:

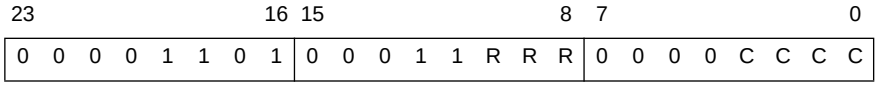
BScc xxxx



BScc xxx



BScc Rn



Instruction Fields:

{cc}	**CCCC**	Condition code (see Table A-43 on page A-251)
{xxxx}		24-bit PC Relative Long Displacement
{xxx}	**aaaaaaaaa**	Signed PC Relative Short Displacement
{Rn}	**RRR**	Address register [R0-R7]

A-6.18 Branch to Subroutine if Bit Clear (BSCLR)

BSCLR BSCLR

Branch to Subroutine if Bit Clear

Operation:	Assembler Syntax:	
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR	#n,[X or Y]:ea,xxxx
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR	#n,[X or Y],aa,xxxx
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR	#n,[X or Y]:pp,xxxx
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR	#n,[X or Y]:qq,xxxx
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR	#n,S,xxxx

Description: The nth 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 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; if the condition is true, the push operation will write over the stack level where the SSH value was 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							

✔ This bit is changed according to the standard definition

× This bit is 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																							

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{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 A-22 on page A-243)

BSET BSET

Bit Test and Set

Operation:		Assembler Syntax:	
D[n] → C	1→ D[n]	BSET	#n,[XorY]:ea
D[n] → C	1→ D[n]	BSET	#n,[XorY]:aa
D[n] → C	1→ D[n]	BSET	#n,[XorY]:pp
D[n] → C	1→ D[n]	BSET	#n,[XorY]:qq
D[n] → C	1→ D[n]	BSET	#n,D

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 condition code 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 which 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). In the data out of the shifter indicates that the accumulator extension register is in use, the instruction will act on the limited value (limited on the maximum positive or negative saturation constant). In addition the “L” flag in the SR will be 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. Not affected otherwise.
- V Set if bit 1 is specified. Not affected otherwise.
- Z Set if bit 2 is specified. Not affected otherwise.
- N Set if bit 3 is specified. Not affected otherwise.
- U Set if bit 4 is specified. Not affected otherwise.
- E Set if bit 5 is specified. Not affected otherwise.
- L Set if bit 6 is specified. Not affected otherwise.
- S Set if bit 7 is specified. Not affected otherwise.

For other destination operands:

- C Set if bit tested is set. Cleared otherwise.
- V Not affected.
- Z Not affected.
- N Not affected.
- U Not affected.
- E Not affected.
- L According to the standard definition.
- S According to the standard definition.

MR Status Bits:

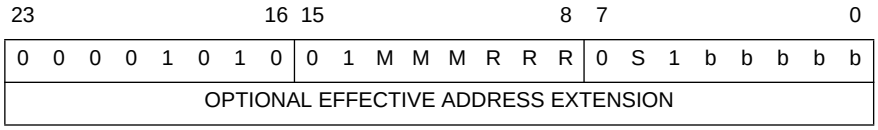
For destination operand SR:

- I0 Changed if bit 8 is specified. Not affected otherwise
- I1 Changed if bit 9 is specified. Not affected otherwise
- So Changed if bit 10 is specified. Not affected otherwise
- S1 Changed if bit 11 is specified. Not affected otherwise
- RM Changed if bit 12 is specified. Not affected otherwise
- SB Changed if bit 13 is specified. Not affected otherwise
- DM Changed if bit 14 is specified. Not affected otherwise
- LF Changed if bit 15 is specified. Not affected otherwise

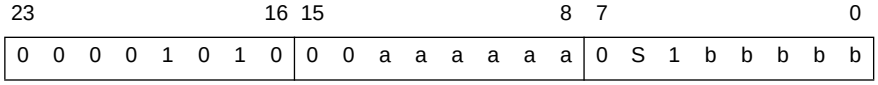
For other destination operands: MR status bits are not affected.

Instruction Formats and opcodes:

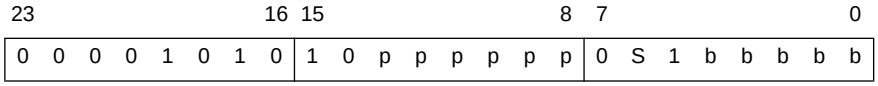
BSET #n,[X or Y]:ea



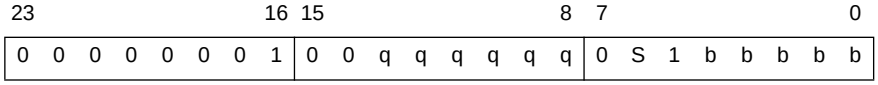
BSET #n,[X or Y]:aa



BSET #n,[X or Y]:pp



BSET #n,[X or Y]:qq



BSET #n,D

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-16 on page A-241)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{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 A-22 on page A-243)

BSR BSR

Branch to Subroutine

Operation:	Assembler Syntax:
PC →SSH;SR →SSL;PC+xxxx→PC	BSR xxxx
PC →SSH;SR →SSL;PC+xxx→PC	BSR xxx
PC →SSH;SR →SSL;PC+Rn→PC	BSR Rn

Description: The address of the instruction immediately following the BSR instruction and the status register are pushed onto the stack. Program execution then continues at location PC+displacement. 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, Long 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.

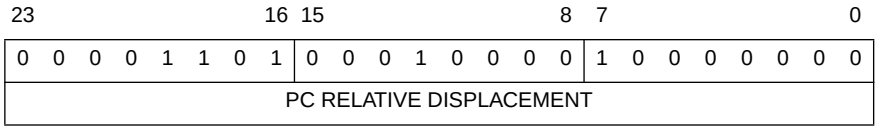
Condition Codes:

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

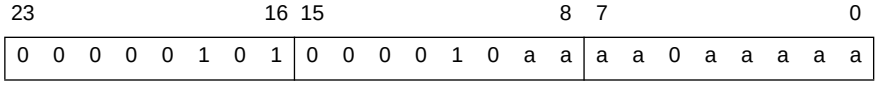
x This bit is unchanged by the instruction

Instruction Formats and opcodes:

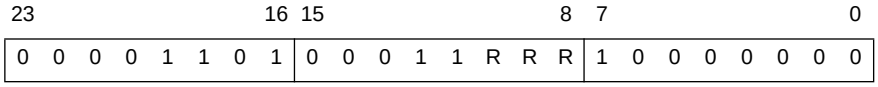
BSR xxxx



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]

A-6.21 Branch to Subroutine if Bit Set (BSSET)

BSSET BSSET

Branch to Subroutine if Bit Set

Operation:	**Assembler Syntax:**	
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET	#n,[X or Y]:ea,xxxx
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET	#n,[X or Y],aa,xxxx
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET	#n,[X or Y]:pp,xxxx
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET	#n,[X or Y]:qq,xxxx
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET	#n,S,xxxx

Description: The nth 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 are 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 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; if the condition is true, the push operation will write over the stack level where the SSH value was 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							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

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

23								16	15							8	7							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								16	15							8	7							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								16	15							8	7							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								16	15							8	7							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								16	15							8	7							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																								

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{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 A-22 on page A-243)

BTST BTST

Bit Test

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

Description: Test the n^{th} bit of the destination operand D. The state of the n^{th} bit is stored in the carry bit C of the condition code register. The bit to be tested is selected by an immediate bit number from 0–23. This instruction is useful for performing serial to parallel conversion when used with the 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. Cleared otherwise.
- ✔ This bit is changed according to the standard definition
- × This bit is unchanged by the instruction

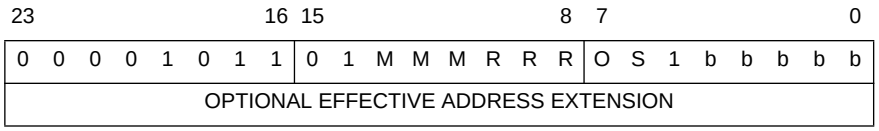
SP — Stack Pointer:

For destination operand SSH: SP — Decrement by 1.

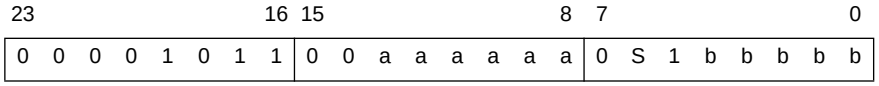
For other destination operands: Not affected

Instruction Formats and opcodes:

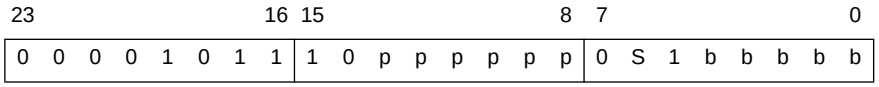
BTST #n,[X or Y]:ea



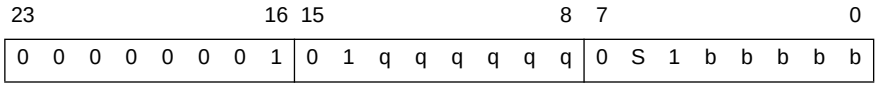
BTST #n,[X or Y]:aa



BTST #n,[X or Y]:pp



BTST #n,[X or Y]:qq



BTST #n,D

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-18 on page A-241)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{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 A-22 on page A-243)

CLB CLB

Count Leading Bits

Operation:

If S[55]=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

Description: Count leading zeros or ones 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 9 minus the number of consecutive leading ones or zeros 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 zeros. The EXP of the destination accumulator D is sign extended.

Notes:

1) If the source accumulator is all zeros then the result will be zero.

2) When 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.

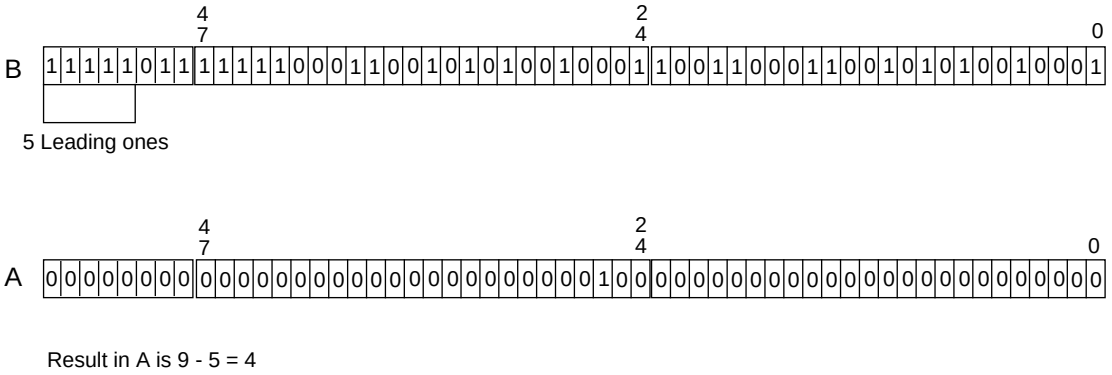
3) This instruction may be used in conjunction with NORMF instruction, to specify the shift direction and amount needed for normalization.

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. Cleared otherwise
- ● Z Set if bits 47-24 of the result are zero.
- ● V Always cleared
- × This bit is unchanged by the instruction

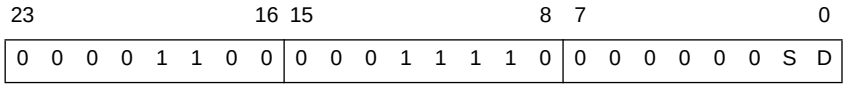
Example: CLB B,A



Result in A is 9 - 5 = 4

Instruction Formats and opcode:

CLB S,D



Instruction Fields:

{D} **D** Destination accumulator [A,B] (see Table A-10 on page A-239)

{S} **S** Source accumulator [A,B] (see Table A-10 on page A-239)

CLR CLR

Clear Accumulator

Operation:

0→ D (parallel move)

Assembler Syntax:

CLR D (parallel move)

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
- ✔ This bit is changed according to the standard definition
- × This bit is 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		

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.25 Compare (CMP)

CMP CMP

Compare

Operation:	**Assembler Syntax:**
S2 – S1 (parallel move)	CMP S1, S2 (parallel move)
S2 – #xx	CMP #xx, S2
S2 – #xxxxxx	CMP #xxxxxx , S2

Description: Subtract the source one operand from the source two accumulator, S2, and update the condition code register. The result of the subtraction operation is not stored.

The source one operand can be a register (word - 24 bits or accumulator - 56 bits), short immediate (6 bits) or long immediate (24 bits). When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the 6 bits will be right aligned and the remaining bits will be zeroed to form a 24-bit source operand.

Note: 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 note particularly applies to the case where it is extended to compare 24-bit operands such as X0 with A1.

Condition Codes:

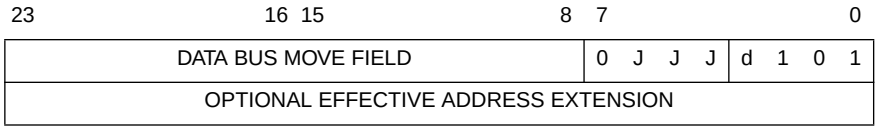
7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition

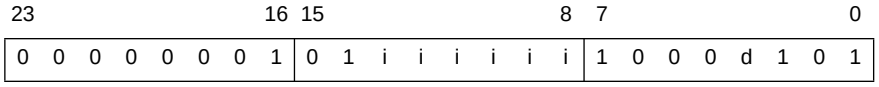
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

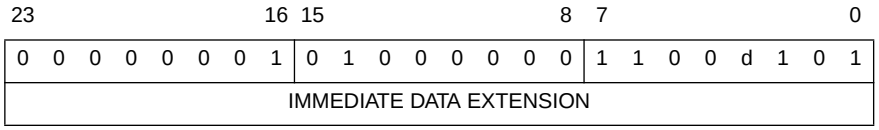
CMP S1, S2



CMP #xx, S2



CMP #xxxxxx,S2



Instruction Fields:

{S1} **JJJ** Source one register [B/A,X0,Y0,X1,Y1] (see Table A-24 on page A-243)
{S2} **d** Source two accumulator [A/B] (see Table A-10 on page A-239)
{#xx} **iiiiii** 6-bit Immediate Short Data
{#xxxxxx} 24-bit Immediate Long Data extension word

CMPM CMPM

Compare Magnitude

Operation:

|S2| – |S1|(parallel move)

Assembler Syntax:

CMPM S1, S2 (parallel move)

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 condition code register. The result of the subtraction operation is not stored.

Note: 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 note 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							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

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		

Instruction Fields:

{S1}	**JJJ**	Source one register [B/A,X0,Y0,X1,Y1] (see Table A-24 on page A-243)
{S2}	**d**	Source two accumulator [A,B] (see Table A-10 on page A-239)

A-6.27 Compare Unsigned (CMPU)

CMPU CMPU

Compare Unsigned

Operation:

S2 – S1

Assembler Syntax:

CMPU S1, S2

Description: Subtract the source one operand, S1, from the source two accumulator, S2, and update the condition code register. The result of the subtraction operation is not stored.

Note: 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 zero
- × This bit is unchanged by the instruction
- ✔ This bit is changed according to the standard definition

Instruction Formats and opcodes:

CMPU S1, S2

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	g	g	g	d

Instruction Fields:

{S1} **ggg** Source one register [A,B,X0,Y0,X1,Y1] (see Table A-15 on page A-240)
{S2} **d** Source two accumulator [A,B] (see Table A-10 on page A-239)

DEBUG DEBUG

Enter Debug Mode

Operation:

Enter the debug mode

Assembler Syntax:

DEBUG

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							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

DEBUG

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	0	0

Instruction Fields: None

A-6.29 Enter Debug Mode Conditionally (DEBUGcc)

DEBUGcc DEBUGcc

Enter Debug Mode Conditionally

Operation:

If cc, then enter the debug mode

Assembler Syntax:

DEBUGcc

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 A-42 on page A-250.

Condition Codes:

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

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

DEBUGcc

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	0	0	0	0	0	0	0	1	1	0	0	0	0	C	C	C	C

Instruction Fields:

{cc} **CCCC** Condition code (see Table A-43 on page A-251)

DEC DEC

Decrement by One

Operation:

D - 1→ D

Assembler Syntax:

DEC D

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
x	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition

x This bit is unchanged by the instruction

Instruction Formats and opcodes:

DEC D

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

Instruction Fields:

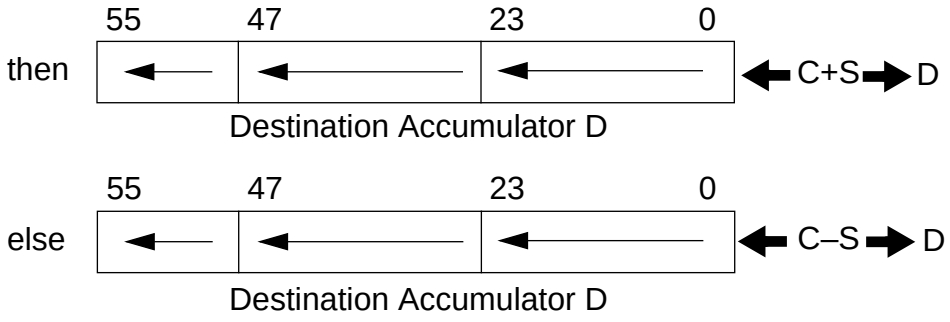
{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.31 Divide Iteration (DIV)

DIV DIV

Divide Iteration

Operation: If D[55]⊕S[23]=1,



where ⊕ denotes the logical exclusive OR operator

Assembler Syntax: DIV S,D

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 which has been sign extended to 56-bits and is stored in the full 56-bit destination accumulator D. The 24-bit divisor is a signed fraction and is stored in the source operand S.** Each DIV iteration calculates one quotient bit using a nonrestoring fractional division algorithm (see description on the next page). After the execution of the first DIV instruction, 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 LS bit 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.** Note that this condition ensures that the magnitude of the quotient is less than one (i.e., is fractional) and precludes division by zero.

The DIV instruction calculates one quotient bit based on the divisor and the previous

partial remainder. To produce an N-bit quotient, the DIV instruction is executed N times where N is the number of bits of precision desired in the quotient, 1;leN;le24. Thus, for a full-precision (24 bit) quotient, 24 DIV iterations are required. In general, executing the DIV instruction N times produces an N-bit quotient and a 48-bit remainder which has (48–N) bits of precision and whose N MS bits are zeros. The partial remainder is not a true remainder and must be corrected due to the nonrestoring nature of the division algorithm before it may 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.

The DIV instruction uses a nonrestoring fractional division algorithm which consists of the following operations (see the previous **Operation** diagram):

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 55-bit destination accumulator D is shifted one bit to the left. The carry bit C is moved into the LS bit (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 MSP portion of the destination accumulator (A1 or B1), and the result is stored back into the MSP portion of that destination accumulator. If the result of the exclusive OR operation previously described was a “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 a “0” (i.e., the sign bits were the same), the source operand S is subtracted from the accumulator. Due to the automatic sign extension of the 24-bit signed divisor, the addition or subtraction operation correctly sets the carry bit C of the condition code register with the next quotient bit.

For extended precision division (i.e., for N-bit quotients where N>24), the DIV instruction is no longer applicable, and a user-defined N-bit division routine is required. For further information on division algorithms, refer to 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 overflow bit V is set
- ● V Set if the MS bit 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.
- × This bit is unchanged by the instruction

Instruction Formats and opcodes:

DIV S,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	J	J	d	0	0	0

Instruction Fields:

{S} **JJ** Source input register [X0,X1,Y0,Y1] (see Table A-12 on page A-239)

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

DMAC DMAC

Double (Multi) Precision Multiply Accumulate with Right Shift

Operation:

[D>>24]±S1∗S2→D
(S1 signed, S2 signed)

[D>>24]±S1∗S2→D
(S1 signed, S2 unsigned)

[D>>24]±S1∗S2→D
(S1 unsigned, S2 unsigned)

Assembler Syntax:

DMACss (±)S1,S2,D (no parallel move)

DMACsu (±)S2,S1,D (no parallel move)

DMACuu (±)S2,S1,D (no parallel move)

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 multiprecision multiplication support.

Condition Codes:

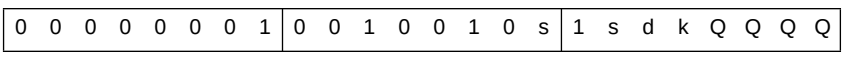
7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

DMAC (±)S1,S2,D



Instruction Fields:

{S1,S2}	**QQQQ**	Source registers S1,S2 [all combinations of X0,X1,Y0 and Y1] (see Table A-30 on page A-245)
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{±±}	**k**	Sign [+,-] (see Table A-29 on page A-244)
{ss,su,uu}	**ss**	[ss,su,uu] (see Table A-39 on page A-248)

DO DO

Start Hardware Loop

Operation:	Assembler Syntax:	
SP+1 → SP;LA → SSH;LC → SSL;[X or Y]:ea → LC SP+1 → SP;PC → SSH;SR → SSL;expr –1 → LA 1 → LF	DO	[Xor Y]:ea,expr
SP+1 → SP;LA → SSH;LC → SSL;[Xor Y]:aa → LC SP+1 → SP;PC → SSH;SR → SSL;expr –1 → LA 1 → LF	DO	[Xor Y]:aa,expr
SP+1 → SP;LA → SSH;LC → SSL;#xxx → LC SP+1 → SP;PC → SSH;SR → SSL;expr –1 → LA 1 → LF	DO	#xxx,expr
SP+1 → SP;LA → SSH;LC → SSL;S → LC SP+1 → SP;PC → SSH;SR → SSL;expr –1 → LA 1 → LF	DO	S,expr

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

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 instruction's source operand is then loaded into the loop counter (LC) register. The LC register contains the remaining number of times the DO loop will be executed and can be accessed from inside the DO loop subject to certain restrictions. If LC initial value is zero and the 16-bit compatibility mode bit (bit 13, SC, in the Chip Status Register) is cleared, the DO loop is not executed. If LC initial value is zero but SC is set, the DO loop will be executed 65,536 times. All address register indirect addressing modes may be used to generate the effective address of the source operand. If immediate short data is specified, the 12 LS bits of LC are loaded with the 12-bit immediate value, and the 12 MS bits of LC 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 which permits the nesting of DO loops. The DO instruction's destination operand (shown as "expr") is then loaded into the loop address (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 program counter (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 (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) is set. This results in the PC being repeatedly compared with LA to determine if the last instruction in the loop has been fetched. If LA equals PC, the last instruction in the loop has been fetched and the loop counter (LC) is tested. If LC is not equal to one, it is decremented by one and SSH is loaded into the PC to fetch the first instruction in the loop again. If LC equals one, the "end-of-loop" processing begins.

When executing a DO loop, 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.

Note: 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.

During the "end-of-loop" processing, the loop flag (LF) from the lower portion (SSL) of SP is written into the status register (SR), the contents of the loop address (LA) register are restored from the upper portion (SSH) of (SP–1), the contents of the loop counter (LC) are restored from the lower portion (SSL) of (SP–1) and the stack pointer (SP) is decremented by two. Instruction fetches now continue at the address of the instruction following the last instruction in the DO loop. Note that LF is the only bit in the status register (SR) that is restored after a hardware DO loop has been exited.

Note: 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 2]
- × This bit is 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		

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{expr}		24-bit Absolute Address in 24-bit extension word
{aa}	**aaaaaa**	Absolute Address [0-63]
{#xxx}	**hhhhiiiiiii**	Immediate Short Data [0-4095]
{S}	**DDDDDD**	Source register [all on-chip registers, **except SSH**] (see Table A-22 on page A-243)

Note:

For DO SP, expr — The actual value that will be loaded into the loop counter (LC) is the value of the stack pointer (SP) before the execution of the DO instruction, incremented by 1.

Thus, if SP=3, the execution of the DO SP,expr instruction will load the loop counter (LC) with the value LC=4.

For DO SSL, expr — The loop counter (LC) will be loaded with its previous value which was saved on the stack by the DO instruction itself.

DO FOREVER DO FOREVER

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

Description: Begin a hardware DO loop that is to be repeated for ever and whose range of execution is terminated by the destination operand (shown above as “expr”). No overhead other than the execution of this DO FOREVER instruction is required to set up this loop. DO FOREVER loops can be nested. 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 loop counter (LC) register is pushed onto the stack but is not updated by this instruction.

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. Stacking the LA, LC, PC, and SR registers permits nesting DO FOREVER loops. The DO FOREVER instruction’s destination operand (shown as “expr”) is then loaded into the Loop Address (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 Program Counter (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 (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. This results in the PC being repeatedly compared with LA to determine if 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 loaded 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. This register can be used by the programer to count the number of loops already executed.

When executing a DO FOREVER loop, the instructions are actually fetched each time through the loop. Therefore, a DO FOREVER 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.

Note: 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.

The loop counter (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
x	x	x	x	x	x	x	x
CCR							

x This bit is 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																								

Instruction Fields: None.

DOR DOR

Start PC relative Hardware Loop

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

Description:

This instruction initiates the beginning of a PC relative hardware program loop. The current 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 which is loaded into the loop counter (LC). The DO loop is executed LC times. If LC initial value is zero and the 16-bit compatibility mode bit (bit 13, SC, in the Chip Status Register) is cleared, the DO loop is not executed. If LC initial value is zero but SC is set, the DO loop will be executed 65,536 times. All address register indirect addressing modes (less Long Displacement) may be used. Register Direct addressing mode may 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 being executed in a hardware loop are interruptible and may 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 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 is at fetch time and ahead of the end of loop execution, instructions which change program flow or change the system stack may not 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 will generate a warning if the restricted instructions are found within their restricted boundaries.

Implementation Notes:

DOR SP,xxxx The actual value that will be loaded in the LC is the value of the SP before the DOR instruction incremented by one.

DOR SSL,xxxx The LC will be loaded with its previous value that was saved in the stack by the DOR instruction itself.

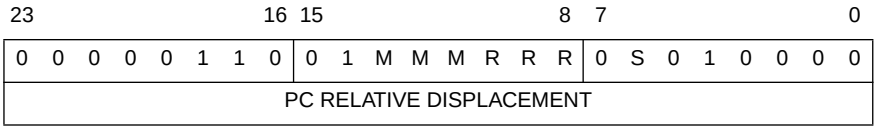
Condition Codes:

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

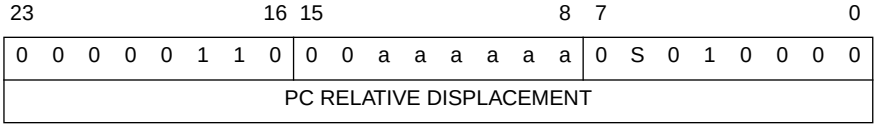
- ● S Set if the instruction sends A/B accumulator contents to XDB or YDB.
- ● L Set if data limiting occurred
- × This bit is unchanged by the instruction

Instruction Formats and opcodes:

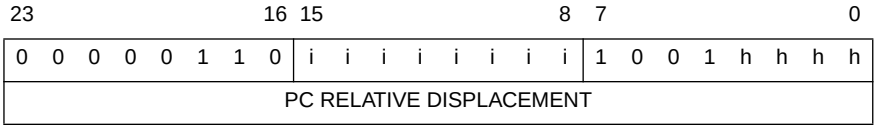
DOR [X or Y]:ea,label



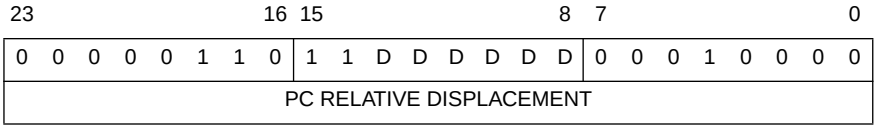
DOR [X or Y]:aa,label



DOR #xxx, label



DOR S, label



Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{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 A-22 on page A-243)

DOR FOREVER DOR FOREVER

Start PC Relative Infinite Loop

Operation:

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

Assembler Syntax:

DOR FOREVER,label

Description: Begin a hardware DO loop that is to be repeated for ever and whose range of execution is terminated by the destination operand (shown above as 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 current 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 by this instruction.

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. Stacking the LA, LC, PC, and SR registers permits nesting DOR FOREVER loops. The DOR FOREVER instruction's destination operand (shown as label) is then loaded into the Loop Address (LA) register after having been added to the PC. This 24-bit operand is located 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. This results in the PC being repeatedly compared with LA to determine if 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. This register can be used by the programer to count the number of loops already executed.

When executing a DOR FOREVER loop, the instructions are actually 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 are not allowed to be equal. The assembler generates an error message when DOR FOREVER loops are improperly nested.

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 loop counter (LC) register is never tested by the DOR 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 DOR FOREVER loop has been executed. If the programer wants to initialize LC to a particular value before the DOR 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 DOR FOREVER loop.

Condition Codes:

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

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

DOR FOREVER

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	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
PC RELATIVE DISPLACEMENT																							

Instruction Fields: None.

ENDDO ENDDO

End Current DO Loop

Operation:

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

Assembler Syntax:

ENDDO

Description: Terminate the current hardware DO loop before the current loop counter (LC) equals one. If the value of the current DO loop counter (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 loop counter (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
x	x	x	x	x	x	x	x
CCR							

x This bit is 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

Instruction Fields: None

A-6.38 Logical Exclusive OR (EOR)

EOR EOR

Logical Exclusive OR

Operation:

S ⊕ D[47:24]→D[47:24] (parallel move)

#xx ⊕ D[47:24]→D[47:24]

#xxxxxx ⊕ D[47:24]→D[47:24]

where ⊕ denotes the logical XOR operator

Assembler Syntax:

EOR S,D (parallel move)

EOR #xx,D

EOR #xxxxxx,D

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 using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the 6 bits will be right aligned and the remaining bits will be 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 zero
- ● V Always cleared
- ✔ This bit is changed according to the standard definition
- × This bit is unchanged by the instruction