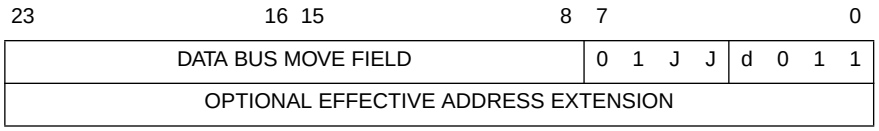
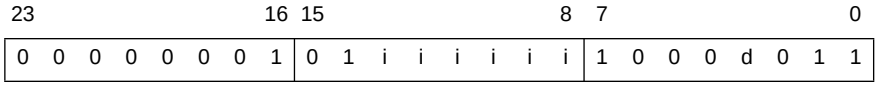
**Instruction Formats and opcodes**:

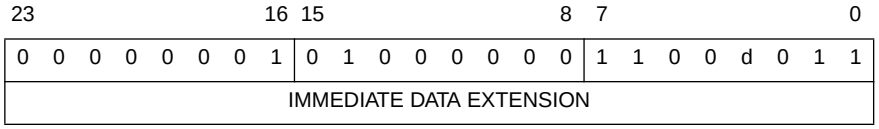
EOR S,D




EOR #xx,D




EOR #xxxxxx,D




**Instruction Fields**:

| | | |
|---|---|---|
| **{S}** | **JJ** | Source 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) |
| **{#xx}** | **iiiiii** | 6-bit Immediate Short Data |
| **{#xxxxxx}** | | 24-bit Immediate Long Data extension word |

# EXTRACT EXTRACT

## Extract Bit Field

**Operation:**

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

S2[(offset+width-1):offset] → D[(width-1):0]
S2[offset+width-1] → D[55:width] (sign extension)

**Assembler Syntax:**

EXTRACT S1,S2,D

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

S2[(offset+width-1):offset] → D[(width-1):0]
S2[offset+width-1] → D[55:width] (sign extension)

EXTRACT #CO,S2,D

**Description:** Extract a bit-field from source accumulator S2. The bit-field width is specified by bits 17-12 in S1 register or in immediate control word #CO. The offset from the least significant bit is specified by bits 5-0 in S1 register or in immediate control word #CO. The extracted field is placed in the destination accumulator D, aligned to the right. The construction of the control register can be done by using the MERGE instruction.

This 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.

Notes:

1) In 16 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 16 bit arithmetic mode, when the width value is zero, then the result will be undefined.

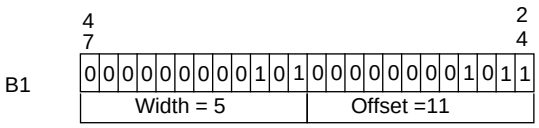
3) If offset + width exceeds the value of 56, the result will be undefined.

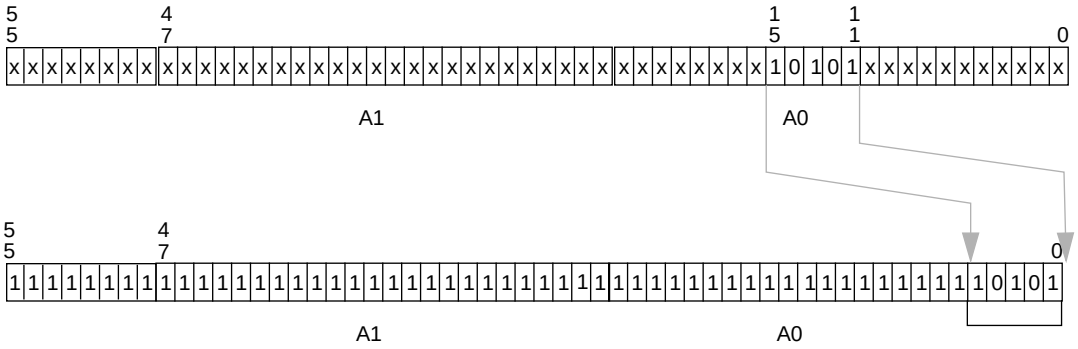
**Condition Codes**:

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

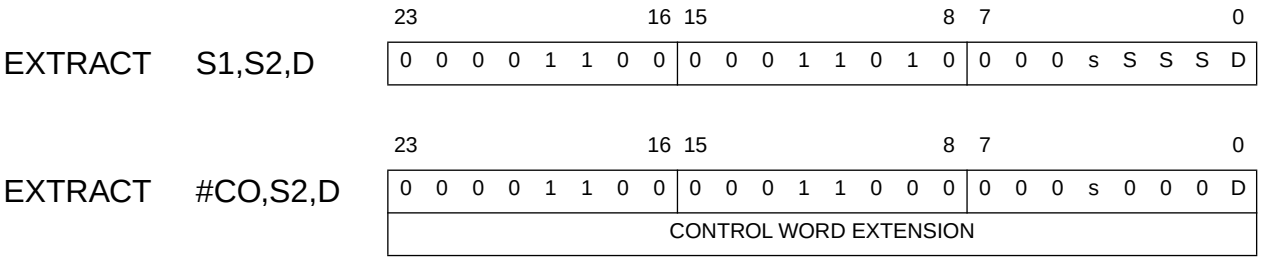
- ● V Always cleared
- ● C Always cleared
- × This bit is unchanged by the instruction
- ✔ This bit is changed according to the standard definition

**Example:** EXTRACT B1,A,A




**Instruction Formats and opcodes**:



**Instruction Fields**:

| | | |
|---|---|---|
| **{S2}** | **s** | Source accumulator [A,B] (see Table A-10 on page A-239) |
| **{D}** | **D** | Destination accumulator [A,B] (see Table A-10 on page A-239) |
| **{S1}** | **SSS** | Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240) |
| **{#CO}** | | Control word extension. |

# 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 → D[55:width]

**Assembler Syntax:**

EXTRACTU S1,S2,D

EXTRACTU #CO,S2,D

**Description:** Extract an unsigned bit-field from source accumulator S2. The bit-field width is specified by bits 17-12 in S1 register or in immediate control word #CO. The offset from the least significant bit is specified by bits 5-0 in S1 register or in immediate control word #CO. The extracted field is placed in the destination accumulator D, aligned to the right. The construction of the control register can be done by using the MERGE instruction.

This is a 56 bits operation. Bits outside the field are filled with zeros.

Notes:

1) in 16 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.

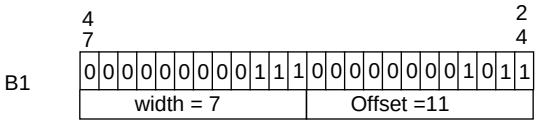
2) If offset + width exceeds the value of 56, the result will be undefined.

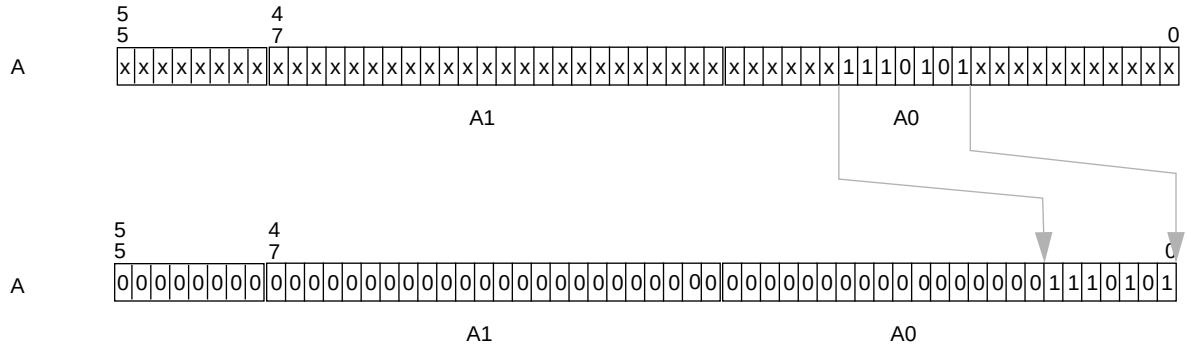
**Condition Codes**:

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

- ● V Always cleared
- ● C Always cleared
- × This bit is unchanged by the instruction
- ✔ This bit is 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 | | | | | | | | | | | | | | | | | | | | | | | | |

**Instruction Fields**:

| | | |
|---|---|---|
| **{S2}** | **s** | Source accumulator [A,B] (see Table A-10 on page A-239) |
| **{D}** | **D** | Destination accumulator [A,B] (see Table A-10 on page A-239) |
| **{S1}** | **SSS** | Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240) |
| **{#CO}** | | Control word extension. |

# IFcc IFcc

## Execute Conditionally without CCR Update

**Operation:**
If cc, then opcode operation

**Assembler Syntax:**
Opcode-Operands IFcc

**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 by using IFcc are the arithmetic and logical instructions that are considered as “parallel” instructions. See Table A-3 and Table A-4 for a list of those instructions.

The conditions that the term “**cc**” may 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**:

IFcc

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 ... 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | C | C | C | C | INSTRUCTION OPCODE |

**Instruction Fields**:

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

# IFcc.U IFcc.U

## Execute Conditionally with CCR Update

**Operation:**
If cc, then opcode operation

**Assembler Syntax:**
Opcode-Operands IFcc

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 by using IFcc.U are the arithmetic and logical instructions that are considered as “parallel” instructions. See Table A-3 and Table A-4 for a list of those instructions.

The conditions that the term “**cc**” may specify are listed on Table A-42 on page A-250

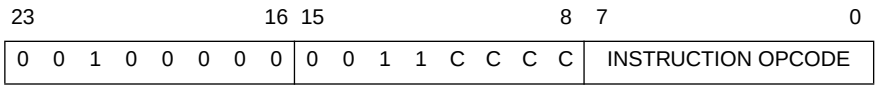
**Condition Codes**:

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

- If the specified condition is true changed according to the instruction. Not changed otherwise.

**Instruction Formats and opcodes**:

IFcc.U




**Instruction Fields**:

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

### A-6.43 Illegal Instruction Interrupt (ILLEGAL)

# ILLEGAL ILLEGAL

## Illegal Instruction Interrupt

**Operation:**
Begin Illegal Instruction exception processing

**Assembler Syntax:**
Opcode-Operands IFcc

**Description:** The ILLEGAL instruction is executed 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 will be 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, etc. at LA are restricted. Clearly restrictions cannot be imposed on illegal instructions.

Since REP is uninterruptable, repeating an ILLEGAL instruction results in the interrupt not being initiated until after completion of the REP. After servicing the interrupt, program control will return 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.

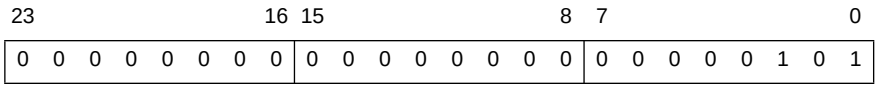
**Condition Codes**:

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

✕ This bit is unchanged by the instruction

**Instruction Formats and opcodes**:

ILLEGAL




**Instruction Fields**: None

# INC INC

## Increment by One

**Operation:**

D +1→ D

**Assembler Syntax:**

INC D

**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 | | | | | | | |

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

**Instruction Formats and opcodes**:

INC 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 | 0 | d |

**Instruction Fields**:

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

### A-6.45 Insert Bit field (INSERT)

# INSERT INSERT

## Insert Bit Field

**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

**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 least significant bit of the S2 register. This bit-field is inserted in the destination accumulator D, with an offset according to bits 5-0 in S1 register. S1 operand can be an immediate control word #CO. Width specified by S1 should not exceed 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.

Notes:

1) In 16 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 value of 16.

2) In 16 bit arithmetic mode, the offset value, located in the offset field, should be the needed offset pre-incremented by the user by a bias of 16.

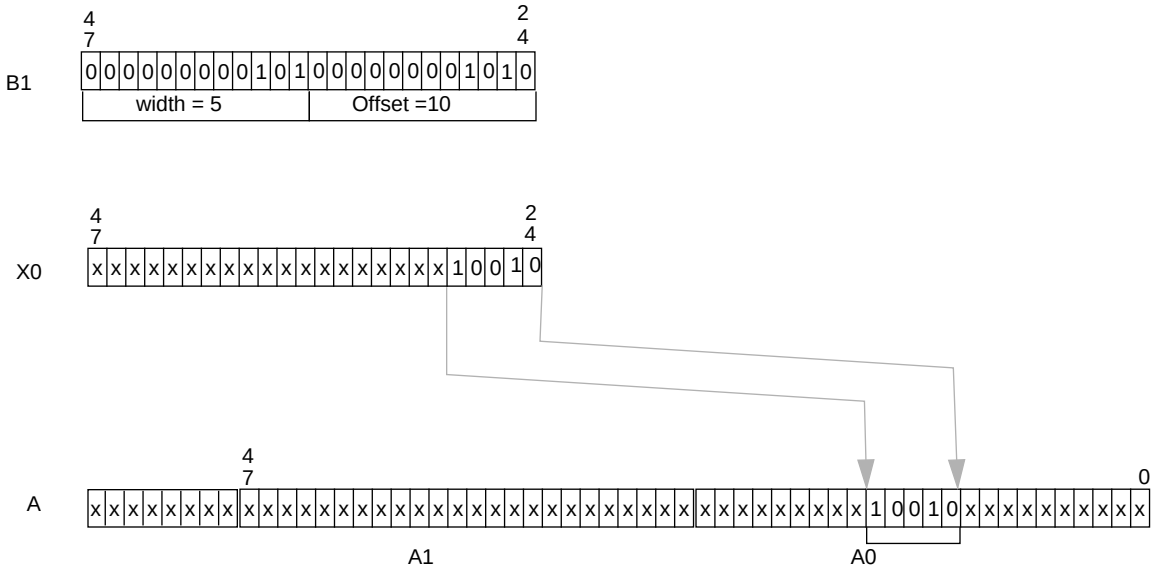
2) If offset + width exceeds the value of 56, the result will be undefined.

**Condition Codes**:

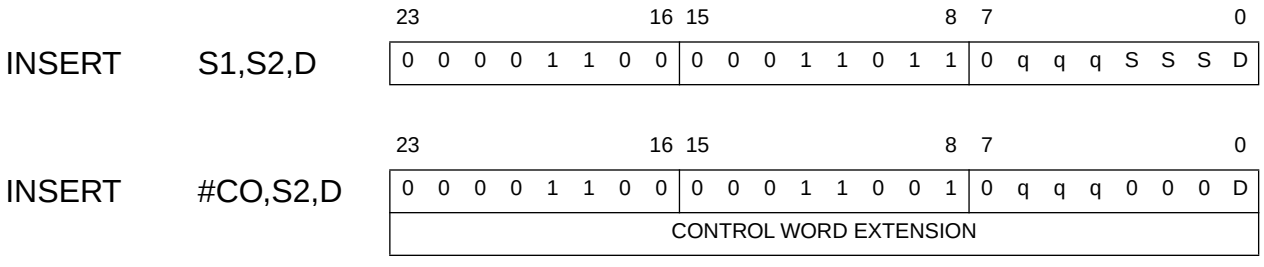
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| S | L | E | U | N | Z | V | C |
| × | × | ✔ | ✔ | ✔ | ✔ | ● | ● |
| CCR | | | | | | | |

- ● V Always cleared
- ● C Always cleared
- × This bit is unchanged by the instruction
- ✔ This bit is changed according to the standard definition

**Example:** INSERT B1,X0,A




**Instruction Formats and opcodes**:



**Instruction Fields**:

| | | |
|---|---|---|
| **{D}** | **D** | Destination accumulator [A,B] (see Table A-10 on page A-239) |
| **{S1}** | **SSS** | Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240) |
| **{S2}** | **qqq** | Source register [X0,X1,Y0,Y1,A0,B0] (see Table A-15 on page A-240) |
| **{#CO}** | | Control word extension. |

# Jcc Jcc

## Jump Conditionally

**Operation:**

If cc, then 0xxx →PC
else PC+1 →PC

If cc, then ea →PC
else PC+1 →PC

**Assembler Syntax:**

Jcc xxx

Jcc ea

**Description:** Jump to the location in program memory given by the instruction's effective address if the specified condition is true. If the specified condition is false, the program counter (PC) is incremented and the effective address is ignored. However, the address register specified in the effective address field is always updated independently of the specified condition. All memory alterable addressing modes may be used for the effective address. A Fast Short Jump addressing mode may also be used. The 12-bit data is zero extended to form the effective address.

The conditions that the term "**cc**" can specify are listed on Table A-42 on page A-250.

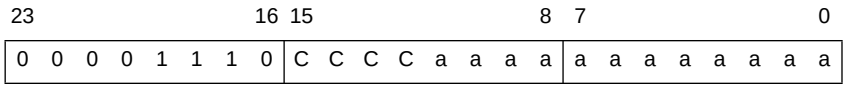
**Condition Codes**:

| 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 | | | | | | | |

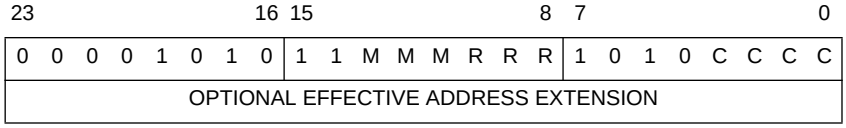
x This bit is unchanged by the instruction

**Instruction Formats and opcodes**:

Jcc xxx




Jcc ea




**Instruction Fields**:

| | | |
|---|---|---|
| **{cc}** | **CCCC** | Condition code (see Table A-43 on page A-251) |
| **{xxx}** | **aaaaaaaaaaaa** | Short Jump Address |
| **{ea}** | **MMMRRR** | Effective Address (see Table A-18 on page A-241) |

### A-6.47 Jump if Bit Clear (JCLR)

# JCLR JCLR

## Jump if Bit Clear

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

**Description:** Jump to the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the $n^{th}$ bit of the source operand S is clear. The bit to be tested is selected by an immediate bit number from 0–23. If the specified memory bit is not clear, the program counter (PC) is incremented and the absolute address in the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the $n^{th}$ bit. All address register indirect addressing modes may be used to reference the source operand S. Absolute Short and I/O Short addressing modes may also be used.

**Condition Codes**:

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

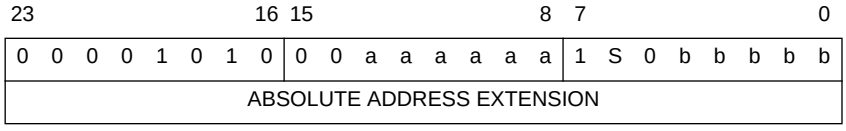
✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

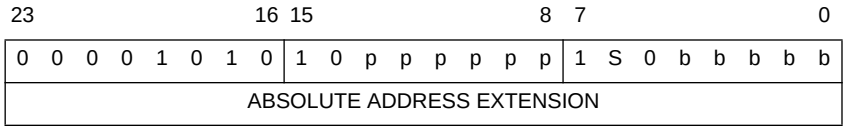
**Instruction Formats and opcodes**:

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

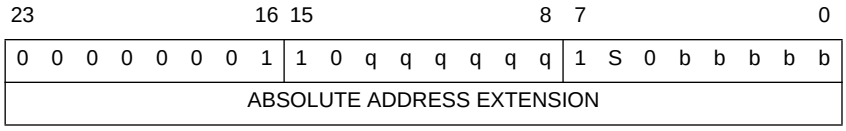

JCLR #n,[X or Y]:aa,xxxx



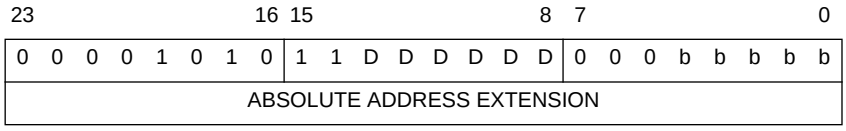

JCLR #n,[X or Y]:pp,xxxx




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




JCLR #n,S,xxxx




**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 absolute Address extension word |
| **{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) |

# JMP JMP

## Jump

**Operation:**

0xxx → Pc

ea→ Pc

**Assembler Syntax:**

JMP xxx

JMP ea

**Description:** Jump to the location in program memory given by the instruction's effective address. All memory alterable addressing modes may be used for the effective address. A Fast Short Jump addressing mode may also be used. The 12-bit data is zero extended to form the effective address.

**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**:

JMP ea

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | | 1 | 1 | M | M | M | R | R | R | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OPTIONAL EFFECTIVE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

JMP xxx

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

**Instruction Fields**:

**{xxx}** **aaaaaaaaaaaa** Short Jump Address
**{ea}** **MMMRRR** Effective Address (see Table A-18 on page A-241)

# JScc JScc

## Jump to Subroutine Conditionally

**Operation:** **Assembler Syntax:**

| | | | | | |
|---|---|---|---|---|---|
| If | cc, | then | SP+1→SP; PC →SSH;SR →SSL;0xxx →PC | JScc | xxx |
| | | else | PC+1→PC | | |
| If | cc, | then | SP+1→SP; PC →SSH;SR →SSL;ea →PC | JScc | ea |
| | | else | PC+1→PC | | |

**Description:** Jump to the subroutine whose location in program memory is given by the instruction's effective address if the specified condition is true. If the specified condition is true, the address of the instruction immediately following the JScc instruction (PC) and the system status register (SR) are pushed onto the system stack. Program execution then continues at the specified effective address in program memory. If the specified condition is false, the program counter (PC) is incremented, and any extension word is ignored. However, the address register specified in the effective address field is always updated independently of the specified condition. All memory alterable addressing modes may be used for the effective address. A fast short jump addressing mode may also be used. The 12-bit data is zero extended to form the effective address.

The conditions that the term "**cc**" can specify are listed on Table A-42 on page A-250.

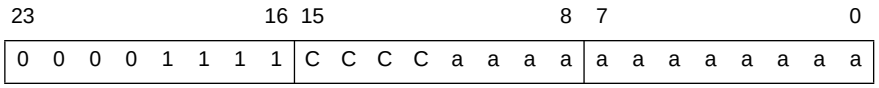
**Condition Codes**:

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

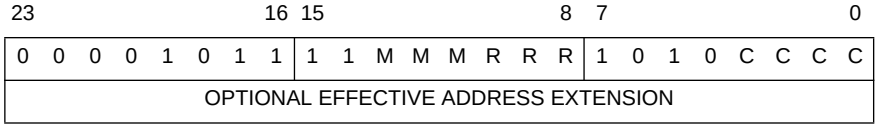
× This bit is unchanged by the instruction

**Instruction Formats and opcodes**:

JScc xxx




JScc ea




**Instruction Fields**:

| | | |
|---|---|---|
| **{cc}** | **CCCC** | Condition code (see Table A-43 on page A-251) |
| **{xxx}** | **aaaaaaaaaaaa** | Short Jump Address |
| **{ea}** | **MMMRRR** | Effective Address (see Table A-18 on page A-241) |

# JSCLR JSCLR

## Jump to Subroutine if Bit Clear

**Operation:**

**Assembler Syntax:**

| | | | |
|---|---|---|---|
| If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL; ;xxxx →PC<br>else PC+1→PC | JSCLR | #n,[X or Y]:ea,xxxx |
| If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL; ;xxxx →PC<br>else PC+1→PC | JSCLR | #n,[X or Y],aa,xxxx |
| If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL; ;xxxx →PC<br>else PC+1→PC | JSCLR | #n,[X or Y]:pp,xxxx |
| If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL; ;xxxx →PC<br>else PC+1→PC | JSCLR | #n,[X or Y]:qq,xxxx |
| If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL; ;xxxx →PC<br>else PC+1→PC | JSCLR | #n,S,xxxx |

**Description:** Jump to the subroutine at the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the $n^{th}$ bit of the source operand S is clear. The bit to be tested is selected by an immediate bit number from 0–23. If the $n^{th}$ bit of the source operand S is clear, the address of the instruction immediately following the JSCLR instruction (PC) and the system status register (SR) are pushed onto the system stack. Program execution then continues at the specified absolute address in the instruction's 24-bit extension word. If the specified memory bit is not clear, the program counter (PC) is incremented and the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the $n^{th}$ bit. All address register indirect addressing modes may be used to reference the source operand S. Absolute short and I/O short addressing modes may also be used·

**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**:

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

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 0 | 1 | M | M | M | R | R | R | 1 | S | 0 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

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

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 0 | 0 | a | a | a | a | a | a | 1 | S | 0 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

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

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 1 | 0 | p | p | p | p | p | p | 1 | S | 0 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

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

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | 1 | 1 | q | q | q | q | q | q | 1 | S | 0 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

JSCLR #n,S,xxxx

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 1 | 1 | D | D | D | D | D | D | 0 | 0 | 0 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

**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 absolute Address extension word |
| **{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) |

### A-6.51 Jump if Bit Set (JSET)

# JSET JSET

## Jump if Bit Set

**Operation:**

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

**Assembler Syntax:**

| | |
|---|---|
| JSET | #n,[X or Y]:ea,xxxx |
| JSET | #n,[X or Y],aa,xxxx |
| JSET | #n,[X or Y]:pp,xxxx |
| JSET | #n,[X or Y]:qq,xxxx |
| JSET | #n,S,xxxx |

**Description:** Jump to the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the $n^{th}$ bit of the source operand S is set. The bit to be tested is selected by an immediate bit number from 0–23. If the specified memory bit is not set, the program counter (PC) is incremented, and the absolute address in the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the $n^{th}$ bit. All address register indirect addressing modes may be used to reference the source operand S. Absolute short and I/O short addressing modes may also be used.

**Condition Codes**:

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

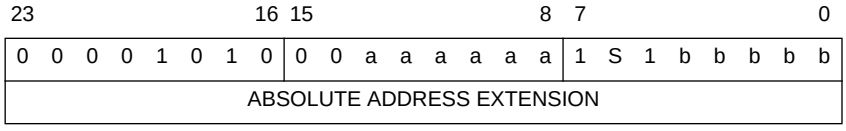
✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

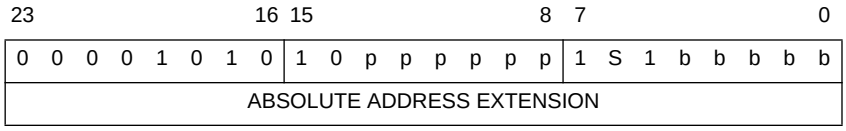
**Instruction Formats and opcodes**:

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

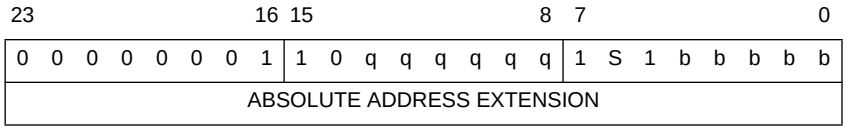

JSET #n,[X or Y]:aa,xxxx



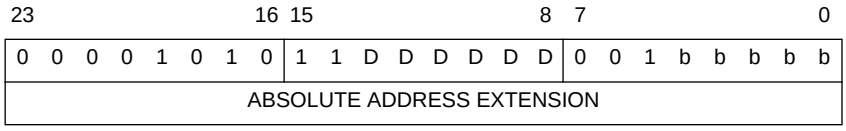

JSET #n,[X or Y]:pp,xxxx




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




JSET #n,S,xxxx




**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 Absolute Address in extension word |
| **{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) |

# JSR JSR

## Jump to Subroutine

**Operation:**

SP+1→SP; PC→SSH; SR→SSL; 0xxx→PC

SP+1→SP; PC→SSH; SR→SSL; ea→PC

**Assembler Syntax:**

JSR xxx

JSR ea

**Description:** Jump to the subroutine whose location in program memory is given by the instruction's effective address. The address of the instruction immediately following the JSR instruction (PC) and the system status register (SR) is pushed onto the system stack. Program execution then continues at the specified effective address in program memory. All memory alterable addressing modes may be used for the effective address. A fast short jump addressing mode may also be used. The 12-bit data is zero extended to form the effective address.

**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**:

JSR ea

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 1 | 1 | M | M | M | R | R | R | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OPTIONAL EFFECTIVE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

JSR xxx

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

**Instruction Fields**:

| | | |
|---|---|---|
| **{xxx}** | **aaaaaaaaaaaa** | Short Jump Address |
| **{ea}** | **MMMRRR** | Effective Address (see Table A-18 on page A-241) |

### A-6.53 Jump to Subroutine if Bit Set (JSSET)

# JSSET JSSET

## Jump to Subroutine if Bit Set

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

**Description:** Jump to the subroutine at the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the $n^{th}$ bit of the source operand S is set. The bit to be tested is selected by an immediate bit number from 0–23. If the $n^{th}$ bit of the source operand S is set, the address of the instruction immediately following the JSSET instruction (PC) and the system status register (SR) are pushed onto the system stack. Program execution then continues at the specified absolute address in the instruction's 24-bit extension word. If the specified memory bit is not set, the program counter (PC) is incremented, and the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the $n^{th}$ bit. All address register indirect addressing modes may be used to reference the source operand S. Absolute short and I/O short addressing modes may also be used.

**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**:

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

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 0 | 1 | M | M | M | R | R | R | 1 | S | 1 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

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

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 0 | 0 | a | a | a | a | a | a | 1 | S | 1 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

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

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 1 | 0 | p | p | p | p | p | p | 1 | S | 1 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

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

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | 1 | 1 | q | q | q | q | q | q | 1 | S | 1 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

JSSET #n,S,xxxx

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 1 | 1 | D | D | D | D | D | D | 0 | 0 | 1 | b | b | b | b | b |
| ABSOLUTE ADDRESS EXTENSION | | | | | | | | | | | | | | | | | | | | | | | | |

**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 absolute Address extension word |
| **{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) |

# LRA LRA

## Load PC Relative Address

**Operation:**

PC+Rn→D

PC+xxxx→D

**Assembler Syntax:**

LRA Rn,D

LRA xxxx,D

**Description:** The PC is added to the specified displacement and the result is stored in destination D. The displacement is a 2's complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Long Displacement and Address Register PC Relative addressing modes may be used. Note that if D is SSH, the SP will be preincremented by one.

**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 opcode**:

LRA Rn,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 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | R | R | R | 0 | 0 | 0 | d | d | d | d | d |

LRA xxxx,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 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | d | d | d | d | d |
| LONG DISPLACEMENT | | | | | | | | | | | | | | | | | | | | | | | |

**Instruction Fields**:

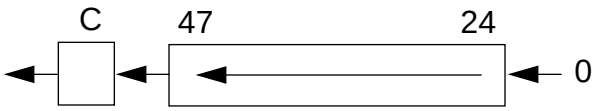
| | | |
|---|---|---|
| **{Rn}** | **RRR** | Address register [R0-R7] |
| **{D}** | **ddddd** | Destination address register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0-R7,N0-N7] (see Table A-31 on page A-245) |
| **{xxxx}** | | 24-bit PC Long Displacement |

### A-6.55 Logical Shift Left (LSL)

# LSL LSL

## Logical Shift Left

**Operation:**




**Assembler Syntax:**

LSL D (parallel move)

LSL #ii,D

LSL S,D

**Description:**

Single-bit shift:

Logically shift bits 47–24 of the destination operand D one bit to the left and store the result in the destination accumulator. Prior to instruction execution, bit 47 of D is shifted into the carry bit C, and a zero is shifted into bit 24 of the destination accumulator D.

Multi-bit shift:

The contents of bits 47-24 of the destination accumulator D are shifted left #ii bits. Bits shifted out of position 47 are lost, but for the last bit which is latched in the carry bit. Zeros are supplied to the vacated positions on the right. The result is placed into bits 47-24 of the destination accumulator D. The number of bits to shift is determined by the 5-bit immediate field in the instruction, or by the unsigned integer located in the control register S. If a zero shift count is specified, the carry bit is cleared.

This is a 24 bit operation. The remaining bits of the destination accumulator are not affected.

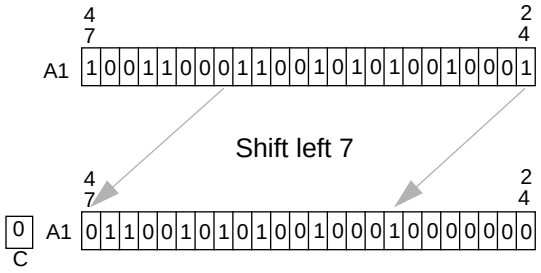
Note: The number of shifts should not exceed the value of 24.

**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
- ● C Set if the last bit shifted out of the operand is set. Cleared otherwise.Cleared for a shift count of zero.
- × This bit is unchanged by the instruction

**Example:** LSL #7, A




**Instruction Formats and opcodes**:

LSL D

| 23 | | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DATA BUS MOVE FIELD | | | | | | | | | 0 | 0 | 1 | 1 | D | 0 | 1 | 1 |
| OPTIONAL EFFECTIVE ADDRESS EXTENSION | | | | | | | | | | | | | | | | |

LSL #ii,D

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | i | i | i | i | i | D |

LSL S,D

| 23 | | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | s | s | s | D |

**Instruction Fields**:

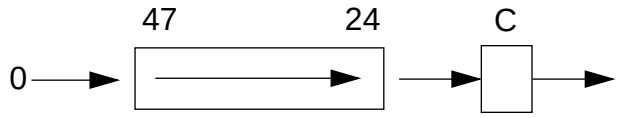
| | | |
|---|---|---|
| **{D}** | **D** | Destination accumulator [A,B] (see Table A-10 on page A-239) |
| **{S}** | **sss** | Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240) |
| **{#ii}** | **iiiii** | 5bit unsigned integer [0-23] denoting the shift amount |

### A-6.56 Logical Shift Right (LSR)

# LSR LSR

## Logical Shift Right

**Operation:**




**Assembler Syntax:**

LSR D (parallel move)

LSR #ii,D

LSR S,D

**Description:**

Single-bit shift:

Logically shift bits 47–24 of the destination operand D one bit to the right and store the result in the destination accumulator. Prior to instruction execution, bit 24 of D is shifted into the carry bit C, and a zero is shifted into bit 47 of the destination accumulator D.

Multi-bit shift:

The contents of bits 47-24 of the destination accumulator D are shifted right #ii bits. Bits shifted out of position 24 are lost, but for the last bit which is latched in the carry bit. Zeros are supplied to the vacated positions on the left. The result is placed into bits 47-24 of the destination accumulator D. The number of bits to shift is determined by the 5-bit immediate field in the instruction, or by the unsigned integer located in the control register S. If a zero shift count is specified, the carry bit is cleared.

This is a 24 bit operation. The remaining bits of the destination register are not affected.

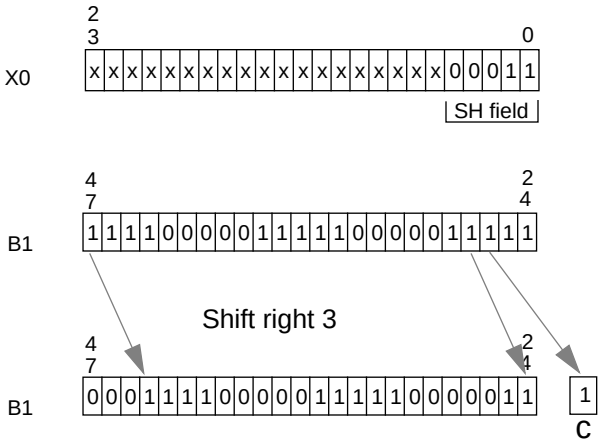
Note: The number of shifts should not exceed the value of 24.

**Condition Codes**:

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

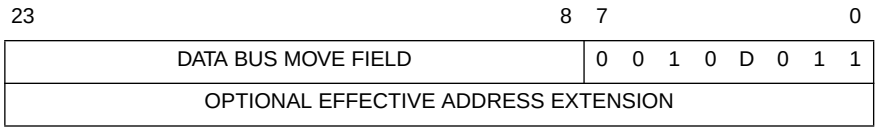
- ● N Set if bit 47 of the result is set
- ● Z Set if bits 47-24 of the result are zero
- ● V Always cleared
- ● C Set if the last bit shifted out of the operand is set. Cleared otherwise. Cleared for a shift count of zero
- x This bit is unchanged by the instruction

**Example:** LSR X0,B

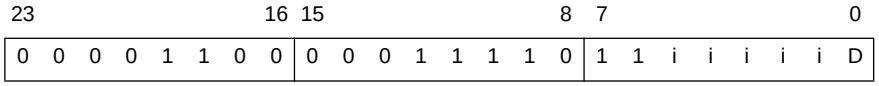


**Instruction Formats and opcodes**:

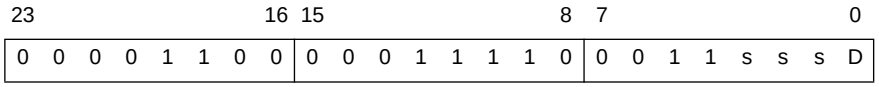
LSR D




LSR #ii,D




LSR S,D




**Instruction Fields**:

**{D}** **D** Destination accumulator [A,B] (see Table A-10 on page A-239)
**{S}** **sss** Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)
**{#ii}** **iiiii** 5 bit unsigned integer [0-23] denoting the shift amount

### A-6.57 Load Updated Address (LUA)

# LUA LUA

## Load Updated address

| **Operation:** | **Assembler Syntax:** |
|---|---|
| ea→D (No update performed) | LUA ea,D |
| Rn+aa→D | LUA (Rn+aa),D |
| ea→D (No update performed) | LEA ea,D |
| Rn+aa→D | LEA (Rn+aa),D |

**Description:** Load the updated address into the destination address register D. The source address register and the update mode used to compute the updated address are specified by the effective address (ea). **Note that the source address register specified in the effective address is not updated. This is the only case where an address register is not updated although stated otherwise in the effective address mode bits.** Only the following addressing modes may be used: Post+N, Post-N, Post+1, Post-1.

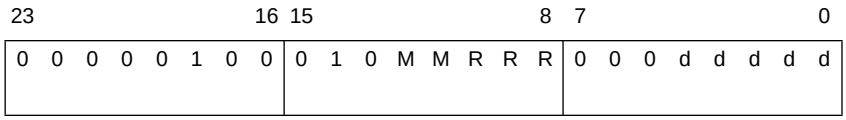
**Condition Codes**:

| 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 | | | | | | | |

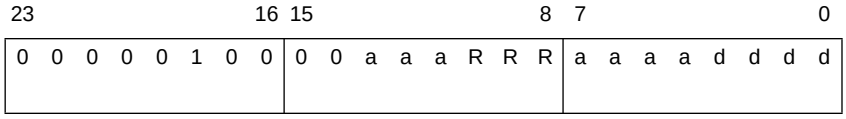
X This bit is unchanged by the instruction

**Instruction Formats and opcode**:

LUA/ LEA ea,D




LUA/ LEA (Rn+aa),D




Note: LEA is a synonym for LUA. The simulator on-line disassembly will translate the opcodes into LUA.

**Instruction Fields**:

| | | |
|---|---|---|
| **{ea}** | **MMRRR** | Effective address (see Table A-20 on page A-242) |
| **{D}** | **ddddd** | Destination address register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0-R7,N0-N7] (see Table A-31 on page A-245) |
| **{D}** | **dddd** | Destination address register [R0-R7,N0-N7] (see Table A-25 on page A-244) |
| **{aa}** | **aaaaaaa** | 7-bit sign extended short displacement address |
| **{Rn}** | **RRR** | Source address register [R0-R7] |

Note: **RRR** refers to a **source** address register (R0-R7), while **dddd/ddddd** refer to a **destination** address register R0-R7 or N0-N7.

# MAC MAC

## Signed Multiply Accumulate

| **Operation:** | **Assembler Syntax:** |
|---|---|
| D±S1∗S2→D (parallel move) | MAC (±)S1,S2,D (parallel move) |
| D±S1∗S2→D (parallel move) | MAC (±)S2,S1,D (parallel move) |
| D±(S1∗$2^{-n}$)→D (**no** parallel move) | MAC (±)S,#n,D (**no** parallel move) |

**Description:** Multiply the two signed 24-bit source operands S1 and S2 (**or** the signed 24-bit source operand S by the positive 24-bit immediate operand $2^{-n}$) and add/subtract the product to/from the specified 56-bit destination accumulator D. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

**Note:** When the processor is in the Double Precision Multiply Mode, the following instructions do not execute in the normal way and should only be used as part of the double precision multiply algorithm:

| | |
|---|---|
| MAC X1, Y0, A | MAC X1, Y0, B |
| MAC X0, Y1, A | MAC X0, Y1, B |
| MAC Y1, X1, A | MAC Y1, X1, B |

**Condition Codes**:

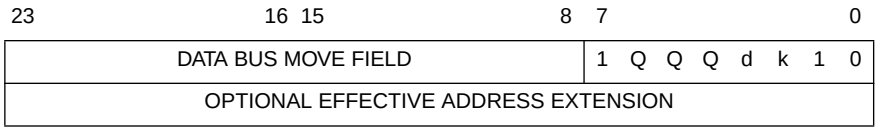
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| S | L | E | U | N | Z | V | C |
| ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | × |
| CCR | | | | | | | |

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

**Instruction Formats and opcodes 1**:

MAC (±)S1,S2,D
MAC (±)S2,S1,D




**Instruction Fields**:

**{S1,S2}** **QQQ** Source registers S1,S2
[X0*X0,Y0*Y0,X1*X0,Y1*Y0,X0*Y1,Y0*X0,X1*Y0,Y1*X1]
(see Table A-26 on page A-244)

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

**{±}** **k** Sign [+,-] (see Table A-29 on page A-244)

**Instruction Formats and opcode 2**:

MAC (±)S,#n,D


**Instruction Fields**:

**{S}** **QQ** Source register [Y1,X0,Y0,X1]] (see Table A-27 on page A-244)

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

**{±}** **k** Sign [+,-] (see Table A-29 on page A-244)

**{#n}** **sssss** Immediate operand (see Table A-32 on page A-246)

# MACI MACI

## Signed Multiply-Accumulate with Immediate Operand

**Operation:**

D±#xxxxxx*S→D

**Assembler Syntax:**

MACI (±)#xxxxxx,S,D

**Description:** Multiply the two signed 24-bit source operands #xxxxxx and S and add/subtract the product to/from the specified 56-bit destination accumulator D. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

**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 opcode:**

MACI (±)#xxxxxx,S,D

| 23 | | | | | | | 16 | 15 | | | | | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | q | q | d | k | 1 | 0 |
| IMMEDIATE DATA EXTENSION | | | | | | | | | | | | | | | | | | | | | | | |

**Instruction Fields:**

| | | |
|---|---|---|
| **{S}** | **qq** | Source register [X0,Y0,X1,Y1] (see Table A-28 on page A-244) |
| **{D}** | **d** | Destination accumulator [A,B] (see Table A-10 on page A-239) |
| **{±}** | **k** | Sign [+,-] (see Table A-29 on page A-244) |
| **#xxxxxx** | | 24-bit Immediate Long Data extension word |

# MAC(su,uu) MAC(su,uu)

## Mixed Multiply Accumulate

**Operation:**

D±S1∗S2→D (S1 unsigned, S2 unsigned)

D±S1∗S2→D (S1 signed, S2 unsigned)

**Assembler Syntax:**

MACuu (±)S1,S2,D (no parallel move)

MACsu (±)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. One or two of the source operands can be unsigned. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

**Condition Codes**:

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

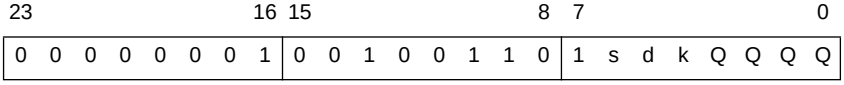
✔ This bit is changed according to the standard definition

✕ This bit is unchanged by the instruction

**Instruction Formats and opcodes**:

MACsu (±)S1,S2,D

MACuu (±)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)

**{s}** [ss,us] (see Table A-40 on page A-249)

### A-6.61 Signed MAC and Round (MACR)

# MACR MACR

## Signed Multiply Accumulate and Round

| **Operation:** | **Assembler Syntax:** | |
|---|---|---|
| D±S1∗S2+r→D (parallel move) | MACR | (±)S1,S2,D (parallel move) |
| D±S1∗S2+r→D (parallel move) | MACR | (±)S2,S1,D (parallel move) |
| D±(S1∗$2^{-n}$)+r→D (**no** parallel move) | MACR | (±)S,#n,D (**no** parallel move) |

**Description:** Multiply the two signed 24-bit source operands S1 and S2 (**or** the signed 24-bit source operand S by the positive 24-bit immediate operand $2^{-n}$), add/subtract the product to/from the specified 56-bit destination accumulator D, and then round the result using either convergent or two's complement rounding. The rounded result is stored in the destination accumulator D.

The "–" sign option negates the specified product prior to accumulation. The default sign option is "+".

The contribution of the LS bits of the result is rounded into the upper portion of the destination accumulator. Once rounding has been completed, the LS bits of the destination accumulator D are loaded with zeros to maintain an unbiased accumulator value which may be reused by the next instruction. The upper portion of the accumulator contains the rounded result which may be read out to the data buses. Refer to the RND instruction for more complete information on the rounding process.

**Condition Codes**:

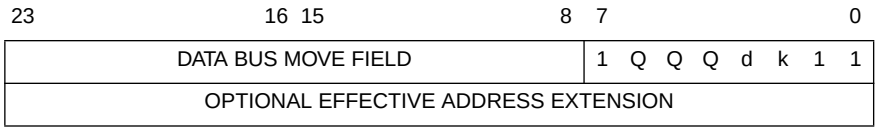
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| S | L | E | U | N | Z | V | C |
| ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | × |
| CCR | | | | | | | |

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

**Instruction Formats and opcodes 1**:

MACR (±)S1,S2,D
MACR (±)S2,S1,D




**Instruction Fields**:

**{S1,S2}** **QQQ** Source registers S1,S2
[X0*X0,Y0*Y0,X1*X0,Y1*Y0,X0*Y1,Y0*X0,X1*Y0,Y1*X1]
(see Table A-26 on page A-244)

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

**{±}** **k** Sign [+,-] (see Table A-29 on page A-244)

**Instruction Formats and opcode 2:**

MACR (±)S,#n,D


**Instruction Fields**:

**{S}** **QQ** Source register [Y1,X0,Y0,X1]] (see Table A-27 on page A-244)

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

**{±}** **k** Sign [+,-] (see Table A-29 on page A-244)

**{#n}** **sssss** Immediate operand (see Table A-32 on page A-246)

# MACRI MACRI

## Signed Multiply-Accumulate and Round with Immediate Operand

**Operation:**

D±#xxxxxx*S→D

**Assembler Syntax:**

MACRI (±)#xxxxxx,S,D

**Description:** Multiply the two signed 24-bit source operands #xxxxxx and S, add/subtract the product to/from the specified 56-bit destination accumulator D, and then round the result using either convergent or two's complement rounding. The rounded result is stored in the destination accumulator D.

The "–" sign option negates the specified product prior to accumulation. The default sign option is "+".

The contribution of the LS bits of the result is rounded into the upper portion of the destination accumulator. Once rounding has been completed, the LS bits of the destination accumulator D are loaded with zeros to maintain an unbiased accumulator value which may be reused by the next instruction. The upper portion of the accumulator contains the rounded result which may be read out to the data buses. Refer to the RND instruction for more complete information on the rounding process.

**Condition Codes**:

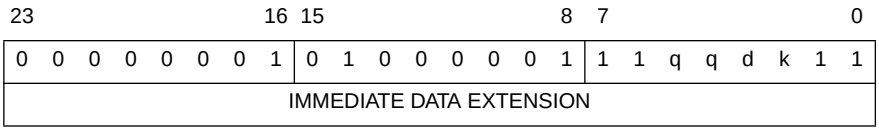
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| S | L | E | U | N | Z | V | C |
| × | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | × |
| CCR | | | | | | | |

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

**Instruction Formats and opcode:**

MACRI (±)#xxxxxx,S,D




**Instruction Fields:**

| | | |
|---|---|---|
| **{S}** | **qq** | Source register [X0,Y0,X1,Y1] (see Table A-28 on page A-244) |
| **{D}** | **d** | Destination accumulator [A,B] (see Table A-10 on page A-239) |
| **{±}** | **k** | Sign [+,-] (see Table A-29 on page A-244) |
| **#xxxxxx** | | 24-bit Immediate Long Data extension word |

# MAX MAX

## Transfer by Signed Value

**Operation:**

If B – A ≤ 0 then A → B

**Assembler Syntax:**

MAX A,B (parallel move)

**Description:** Subtract the signed value of the source accumulator from the signed value of the destination accumulator. If the difference is negative or zero (i.e. A ≥ B) then transfer the source accumulator to destination accumulator, otherwise do not change destination accumulator.

This is a 56 bit operation.

Note: The Carry condition code signifies that a transfer has been performed.

**Condition Codes**:

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

- ● C Cleared if the conditional transfer was performed. Set otherwise.
- ✔ This bit is changed according to the standard definition
- × This bit is unchanged by the instruction

**Instruction Formats and opcodes**:

MAX A, B

| 23 16 | 15 8 | 7 0 |
|---|---|---|
| DATA BUS MOVE FIELD | 0 0 0 1 | 1 1 0 1 |
| OPTIONAL EFFECTIVE ADDRESS EXTENSION | | |