4.6 Instruction Set Description

Table 4-20 shows all available instructions:

Table 4-20. Instruction Map of MSP430X

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx	MOVA, CMPA, ADDA, SUBA, RRCM, RRAM, RLAM, RRUM															
10xx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI	CALLA		
14xx	PUSHM.A, POPM.A, PUSHM.W, POPM.W															
18xx	Extension word for Format I and Format II instructions															
1Cxx																
20xx	JNE, JNZ															
24xx	JEQ, JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

4.6.1 Extended Instruction Binary Descriptions

Detailed MSP430X instruction binary descriptions are shown in the following tables.

Instruction	Instruction Group				src or data.19:16	Instruction Identifier				dst	
	15			12	11 8	7			4	3 0	
MOVA	0	0	0	0	src	0	0	0	0	dst	MOVA @Rsrc,Rdst
	0	0	0	0	src	0	0	0	1	dst	MOVA @Rsrc+,Rdst
	0	0	0	0	&abs.19:16	0	0	1	0	dst	MOVA &abs20,Rdst
	&abs.15:0										
	0	0	0	0	src	0	0	1	1	dst	MOVA x(Rsrc),Rdst
	x.15:0										±15-bit index x
	0	0	0	0	src	0	1	1	0	&abs.19:16	MOVA Rsrc,&abs20
	&abs.15:0										
	0	0	0	0	src	0	1	1	1	dst	MOVA Rsrc,X(Rdst)
	x.15:0										±15-bit index x
	0	0	0	0	imm.19:16	1	0	0	0	dst	MOVA #imm20,Rdst
	imm.15:0										
CMPA	0	0	0	0	imm.19:16	1	0	0	1	dst	CMPA #imm20,Rdst
	imm.15:0										
ADDA	0	0	0	0	imm.19:16	1	0	1	0	dst	ADDA #imm20,Rdst
	imm.15:0										
SUBA	0	0	0	0	imm.19:16	1	0	1	1	dst	SUBA #imm20,Rdst
	imm.15:0										
MOVA	0	0	0	0	src	1	1	0	0	dst	MOVA Rsrc,Rdst
CMPA	0	0	0	0	src	1	1	0	1	dst	CMPA Rsrc,Rdst
ADDA	0	0	0	0	src	1	1	1	0	dst	ADDA Rsrc,Rdst
SUBA	0	0	0	0	src	1	1	1	1	dst	SUBA Rsrc,Rdst

Instruction	Instruction Group				Bit Loc.	Inst. ID		Instruction Identifier				dst	
	15			12	11 10	9	8	7			4	3 0	
RRCM.A	0	0	0	0	n – 1	0	0	0	1	0	0	dst	RRCM.A #n,Rdst
RRAM.A	0	0	0	0	n – 1	0	1	0	1	0	0	dst	RRAM.A #n,Rdst
RLAM.A	0	0	0	0	n – 1	1	0	0	1	0	0	dst	RLAM.A #n,Rdst
RRUM.A	0	0	0	0	n – 1	1	1	0	1	0	0	dst	RRUM.A #n,Rdst
RRCM.W	0	0	0	0	n – 1	0	0	0	1	0	1	dst	RRCM.W #n,Rdst
RRAM.W	0	0	0	0	n – 1	0	1	0	1	0	1	dst	RRAM.W #n,Rdst
RLAM.W	0	0	0	0	n – 1	1	0	0	1	0	1	dst	RLAM.W #n,Rdst
RRUM.W	0	0	0	0	n – 1	1	1	0	1	0	1	dst	RRUM.W #n,Rdst

Instruction	Instruction Identifier												dst				
	15			12	11			8	7	6	5	4	3			0	
RETI	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	
CALLA	0	0	0	1	0	0	1	1	0	1	0	0	dst				`CALLA Rdst`
	0	0	0	1	0	0	1	1	0	1	0	1	dst				`CALLA x(Rdst)`
	x.15:0																
	0	0	0	1	0	0	1	1	0	1	1	0	dst				`CALLA @Rdst`
	0	0	0	1	0	0	1	1	0	1	1	1	dst				`CALLA @Rdst+`
	0	0	0	1	0	0	1	1	1	0	0	0	&abs.19:16				`CALLA &abs20`
	&abs.15:0																
	0	0	0	1	0	0	1	1	1	0	0	1	x.19:16				`CALLA EDE`
	x.15:0																`CALLA x(PC)`
	0	0	0	1	0	0	1	1	1	0	1	1	imm.19:16				`CALLA #imm20`
	imm.15:0																
Reserved	0	0	0	1	0	0	1	1	1	0	1	0	x	x	x	x	
Reserved	0	0	0	1	0	0	1	1	1	1	x	x	x	x	x	x	
PUSHM.A	0	0	0	1	0	1	0	0	n – 1				dst				`PUSHM.A #n,Rdst`
PUSHM.W	0	0	0	1	0	1	0	1	n – 1				dst				`PUSHM.W #n,Rdst`
POPM.A	0	0	0	1	0	1	1	0	n – 1				dst – n + 1				`POPM.A #n,Rdst`
POPM.W	0	0	0	1	0	1	1	1	n – 1				dst – n + 1				`POPM.W #n,Rdst`

4.6.2 MSP430 Instructions

The MSP430 instructions are listed and described on the following pages.

4.6.2.1 ADC

*** ADC[.W]**	Add carry to destination
*** ADC.B**	Add carry to destination
Syntax	`ADC dst` or `ADC.W dst`
	`ADC.B dst`
Operation	dst + C → dst
Emulation	`ADDC #0,dst`
	`ADDC.B #0,dst`
Description	The carry bit (C) is added to the destination operand. The previous contents of the destination are lost.
Status Bits	N: Set if result is negative, reset if positive
	Z: Set if result is zero, reset otherwise
	C: Set if dst was incremented from 0FFFFh to 0000, reset otherwise
	Set if dst was incremented from 0FFh to 00, reset otherwise
	V: Set if an arithmetic overflow occurs, otherwise reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 16-bit counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
ADD     @R13,0(R12)      ; Add LSDs
ADC     2(R12)           ; Add carry to MSD
```

Example The 8-bit counter pointed to by R13 is added to a 16-bit counter pointed to by R12.

```
ADD.B   @R13,0(R12)     ; Add LSDs
ADC.B   1(R12)          ; Add carry to MSD
```

4.6.2.2 ADD

ADD[.W]	Add source word to destination word
ADD.B	Add source byte to destination byte

Syntax	`ADD src,dst` or `ADD.W src,dst`
	`ADD.B src,dst`

Operation	src + dst → dst
Description	The source operand is added to the destination operand. The previous content of the destination is lost.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB of the result, reset otherwise V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Ten is added to the 16-bit counter CNTR located in lower 64 K.

```
ADD.W   #10,&CNTR      ; Add 10 to 16-bit counter
```

Example A table word pointed to by R5 (20-bit address in R5) is added to R6. The jump to label TONI is performed on a carry.

```
ADD.W    @R5,R6        ; Add table word to R6. R6.19:16 = 0
JC       TONI          ; Jump if carry
...                    ; No carry
```

Example A table byte pointed to by R5 (20-bit address) is added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1. R6.19:8 = 0

```
ADD.B    @R5+,R6       ; Add byte to R6. R5 + 1. R6: 000xxh
JNC      TONI          ; Jump if no carry
...                    ; Carry occurred
```

4.6.2.3 ADDC

ADDC[.W]	Add source word and carry to destination word
ADDC.B	Add source byte and carry to destination byte

Syntax	`ADDC src,dst` or `ADDC.W src,dst`
	`ADDC.B src,dst`

Operation	src + dst + C → dst
Description	The source operand and the carry bit C are added to the destination operand. The previous content of the destination is lost.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB of the result, reset otherwise V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Constant value 15 and the carry of the previous instruction are added to the 16-bit counter CNTR located in lower 64 K.

```
ADDC.W    #15,&CNTR     ; Add 15 + C to 16-bit CNTR
```

Example A table word pointed to by R5 (20-bit address) and the carry C are added to R6. The jump to label TONI is performed on a carry. R6.19:16 = 0

```
ADDC.W    @R5,R6        ; Add table word + C to R6
JC        TONI          ; Jump if carry
...                     ; No carry
```

Example A table byte pointed to by R5 (20-bit address) and the carry bit C are added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1. R6.19:8 = 0

```
ADDC.B    @R5+,R6       ; Add table byte + C to R6. R5 + 1
JNC       TONI          ; Jump if no carry
...                     ; Carry occurred
```

4.6.2.4 AND

AND[.W]	Logical AND of source word with destination word
AND.B	Logical AND of source byte with destination byte

Syntax	`AND src,dst` or `AND.W src,dst`
	`AND.B src,dst`

Operation	src .and. dst → dst
Description	The source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if the result is not zero, reset otherwise. C = (.not. Z) V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The bits set in R5 (16-bit data) are used as a mask (AA55h) for the word TOM located in the lower 64 K. If the result is zero, a branch is taken to label TONI. R5.19:16 = 0

```
MOV     #AA55h,R5      ; Load 16-bit mask to R5
AND     R5,&TOM        ; TOM .and. R5 -> TOM
JZ      TONI           ; Jump if result 0
...                    ; Result > 0
```

or shorter:

```
AND     #AA55h,&TOM    ; TOM .and. AA55h -> TOM
JZ      TONI           ; Jump if result 0
```

Example A table byte pointed to by R5 (20-bit address) is logically ANDed with R6. R5 is incremented by 1 after the fetching of the byte. R6.19:8 = 0

```
AND.B   @R5+,R6        ; AND table byte with R6. R5 + 1
```

4.6.2.5 BIC

BIC[.W]	Clear bits set in source word in destination word
BIC.B	Clear bits set in source byte in destination byte
Syntax	`BIC src,dst` or `BIC.W src,dst`
	`BIC.B src,dst`
Operation	(.not. src) .and. dst → dst
Description	The inverted source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.
Status Bits	N: Not affected
	Z: Not affected
	C: Not affected
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The bits 15:14 of R5 (16-bit data) are cleared. R5.19:16 = 0

```
BIC     #0C000h,R5     ; Clear R5.19:14 bits
```

Example A table word pointed to by R5 (20-bit address) is used to clear bits in R7. R7.19:16 = 0

```
BIC.W   @R5,R7         ; Clear bits in R7 set in @R5
```

Example A table byte pointed to by R5 (20-bit address) is used to clear bits in Port1.

```
BIC.B   @R5,&P1OUT     ; Clear I/O port P1 bits set in @R5
```

4.6.2.6 BIS

BIS[.W]	Set bits set in source word in destination word
BIS.B	Set bits set in source byte in destination byte

Syntax	`BIS src,dst` or `BIS.W src,dst`
	`BIS.B src,dst`

Operation	src .or. dst → dst
Description	The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected.
Status Bits	N: Not affected Z: Not affected C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Bits 15 and 13 of R5 (16-bit data) are set to one. R5.19:16 = 0

```
    BIS      #A000h,R5      ; Set R5 bits
```

Example A table word pointed to by R5 (20-bit address) is used to set bits in R7. R7.19:16 = 0

```
    BIS.W   @R5,R7          ; Set bits in R7
```

Example A table byte pointed to by R5 (20-bit address) is used to set bits in Port1. R5 is incremented by 1 afterwards.

```
    BIS.B   @R5+,&P1OUT     ; Set I/O port P1 bits. R5 + 1
```

4.6.2.7 BIT

BIT[.W]	Test bits set in source word in destination word
BIT.B	Test bits set in source byte in destination byte
Syntax	`BIT src,dst` or `BIT.W src,dst`
	`BIT.B src,dst`
Operation	src .and. dst
Description	The source operand and the destination operand are logically ANDed. The result affects only the status bits in SR.
	Register mode: the register bits Rdst.19:16 (.W) resp. Rdst. 19:8 (.B) are not cleared!
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)
	Z: Set if result is zero, reset otherwise
	C: Set if the result is not zero, reset otherwise. C = (.not. Z)
	V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Test if one (or both) of bits 15 and 14 of R5 (16-bit data) is set. Jump to label TONI if this is the case. R5.19:16 are not affected.

```
BIT     #C000h,R5       ; Test R5.15:14 bits
JNZ     TONI            ; At least one bit is set in R5
...                     ; Both bits are reset
```

Example A table word pointed to by R5 (20-bit address) is used to test bits in R7. Jump to label TONI if at least one bit is set. R7.19:16 are not affected.

```
BIT.W   @R5,R7          ; Test bits in R7
JC      TONI            ; At least one bit is set
...                     ; Both are reset
```

Example A table byte pointed to by R5 (20-bit address) is used to test bits in output Port1. Jump to label TONI if no bit is set. The next table byte is addressed.

```
BIT.B   @R5+,&P1OUT     ; Test I/O port P1 bits. R5 + 1
JNC     TONI            ; No corresponding bit is set
...                     ; At least one bit is set
```

4.6.2.8 BR, BRANCH

*** BR, BRANCH**	Branch to destination in lower 64K address space
Syntax	`BR dst`
Operation	dst → PC
Emulation	`MOV dst,PC`
Description	An unconditional branch is taken to an address anywhere in the lower 64K address space. All source addressing modes can be used. The branch instruction is a word instruction.
Status Bits	Status bits are not affected.
Example	Examples for all addressing modes are given.

```
    BR      #EXEC     ; Branch to label EXEC or direct branch (for example #0A4h)
                      ; Core instruction MOV @PC+,PC

    BR      EXEC      ; Branch to the address contained in EXEC
                      ; Core instruction MOV X(PC),PC
                      ; Indirect address

    BR      &EXEC     ; Branch to the address contained in absolute
                      ; address EXEC
                      ; Core instruction MOV X(0),PC
                      ; Indirect address

    BR      R5        ; Branch to the address contained in R5
                      ; Core instruction MOV R5,PC
                      ; Indirect R5

    BR      @R5       ; Branch to the address contained in the word
                      ; pointed to by R5.
                      ; Core instruction MOV @R5,PC
                      ; Indirect, indirect R5

    BR      @R5+      ; Branch to the address contained in the word pointed
                      ; to by R5 and increment pointer in R5 afterwards.
                      ; The next time-S/W flow uses R5 pointer-it can
                      ; alter program execution due to access to
                      ; next address in a table pointed to by R5
                      ; Core instruction MOV @R5,PC
                      ; Indirect, indirect R5 with autoincrement

    BR      X(R5)     ; Branch to the address contained in the address
                      ; pointed to by R5 + X (for example table with address
                      ; starting at X). X can be an address or a label
                      ; Core instruction MOV X(R5),PC
                      ; Indirect, indirect R5 + X
```

4.6.2.9 CALL

CALL	Call a subroutine in lower 64 K
Syntax	`CALL dst`
Operation	dst → tmp 16-bit dst is evaluated and stored SP – 2 → SP PC → @SP updated PC with return address to TOS tmp → PC saved 16-bit dst to PC
Description	A subroutine call is made from an address in the lower 64 K to a subroutine address in the lower 64 K. All seven source addressing modes can be used. The call instruction is a word instruction. The return is made with the RET instruction.
Status Bits	Status bits are not affected. PC.19:16 cleared (address in lower 64 K)
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Examples	Examples for all addressing modes are given.

Immediate Mode: Call a subroutine at label EXEC (lower 64 K) or call directly to address.

```
CALL   #EXEC              ; Start address EXEC
CALL   #0AA04h            ; Start address 0AA04h
```

Symbolic Mode: Call a subroutine at the 16-bit address contained in address EXEC. EXEC is located at the address (PC + X) where X is within PC ± 32 K.

```
CALL   EXEC               ; Start address at @EXEC. z16(PC)
```

Absolute Mode: Call a subroutine at the 16-bit address contained in absolute address EXEC in the lower 64 K.

```
CALL   &EXEC              ; Start address at @EXEC
```

Register mode: Call a subroutine at the 16-bit address contained in register R5.15:0.

```
CALL   R5                 ; Start address at R5
```

Indirect Mode: Call a subroutine at the 16-bit address contained in the word pointed to by register R5 (20-bit address).

```
CALL   @R5                ; Start address at @R5
```

4.6.2.10 CLR

*** CLR[.W]**	Clear destination
*** CLR.B**	Clear destination
Syntax	`CLR dst` or `CLR.W dst`
	`CLR.B dst`
Operation	0 → dst
Emulation	`MOV #0,dst`
	`MOV.B #0,dst`
Description	The destination operand is cleared.
Status Bits	Status bits are not affected.
Example	RAM word TONI is cleared.

```
    CLR     TONI      ; 0 -> TONI
```

Example Register R5 is cleared.

```
    CLR     R5
```

Example RAM byte TONI is cleared.

```
    CLR.B   TONI      ; 0 -> TONI
```

4.6.2.11 CLRC

*** CLRC**	Clear carry bit
Syntax	`CLRC`
Operation	0 → C
Emulation	`BIC #1,SR`
Description	The carry bit (C) is cleared. The clear carry instruction is a word instruction.
Status Bits	N: Not affected
	Z: Not affected
	C: Cleared
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 16-bit decimal counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
CLRC                    ; C=0: defines start
DADD    @R13,0(R12)     ; add 16-bit counter to low word of 32-bit counter
DADC    2(R12)          ; add carry to high word of 32-bit counter
```

4.6.2.12 CLRN

*** CLRN**	Clear negative bit
Syntax	`CLRN`
Operation	0 → N or (.NOT.src .AND. dst → dst)
Emulation	`BIC #4,SR`
Description	The constant 04h is inverted (0FFFBh) and is logically ANDed with the destination operand. The result is placed into the destination. The clear negative bit instruction is a word instruction.
Status Bits	N: Reset to 0 Z: Not affected C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The negative bit in the SR is cleared. This avoids special treatment with negative numbers of the subroutine called.

```
            CLRN
            CALL   SUBR
            ......
            ......
SUBR        JN     SUBRET     ; If input is negative: do nothing and return
            ......
            ......
            ......
SUBRET      RET
```

4.6.2.13 CLRZ

*** CLRZ**	Clear zero bit
Syntax	`CLRZ`
Operation	0 → Z or (.NOT.src .AND. dst → dst)
Emulation	`BIC #2,SR`
Description	The constant 02h is inverted (0FFFDh) and logically ANDed with the destination operand. The result is placed into the destination. The clear zero bit instruction is a word instruction.
Status Bits	N: Not affected Z: Reset to 0 C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The zero bit in the SR is cleared.

```
CLRZ
```

Indirect, Auto-Increment mode: Call a subroutine at the 16-bit address contained in the word pointed to by register R5 (20-bit address) and increment the 16-bit address in R5 afterwards by 2. The next time the software uses R5 as a pointer, it can alter the program execution due to access to the next word address in the table pointed to by R5.

```
CALL   @R5+          ; Start address at @R5. R5 + 2
```

Indexed mode: Call a subroutine at the 16-bit address contained in the 20-bit address pointed to by register (R5 + X); for example, a table with addresses starting at X. The address is within the lower 64KB. X is within ±32KB.

```
CALL   X(R5)         ; Start address at @(R5+X). z16(R5)
```

4.6.2.14 CMP

CMP[.W] Compare source word and destination word

CMP.B Compare source byte and destination byte

Syntax `CMP src,dst` or `CMP.W src,dst`

`CMP.B src,dst`

Operation (.not.src) + 1 + dst

or

dst – src

Emulation `BIC #2,SR`

Description The source operand is subtracted from the destination operand. This is made by adding the 1s complement of the source + 1 to the destination. The result affects only the status bits in SR.

Register mode: the register bits Rdst.19:16 (.W) resp. Rdst. 19:8 (.B) are not cleared.

Status Bits
N: Set if result is negative (src > dst), reset if positive (src = dst)
Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)
C: Set if there is a carry from the MSB, reset otherwise
V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow).

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Compare word EDE with a 16-bit constant 1800h. Jump to label TONI if EDE equals the constant. The address of EDE is within PC + 32 K.

```
CMP     #01800h,EDE      ; Compare word EDE with 1800h
JEQ     TONI             ; EDE contains 1800h
...                      ; Not equal
```

Example A table word pointed to by (R5 + 10) is compared with R7. Jump to label TONI if R7 contains a lower, signed 16-bit number. R7.19:16 is not cleared. The address of the source operand is a 20-bit address in full memory range.

```
CMP.W   10(R5),R7        ; Compare two signed numbers
JL      TONI             ; R7 < 10(R5)
...                      ; R7 >= 10(R5)
```

Example A table byte pointed to by R5 (20-bit address) is compared to the value in output Port1. Jump to label TONI if values are equal. The next table byte is addressed.

```
CMP.B   @R5+,&P1OUT      ; Compare P1 bits with table. R5 + 1
JEQ     TONI             ; Equal contents
...                      ; Not equal
```

4.6.2.15 DADC

*** DADC[.W]**	Add carry decimally to destination
*** DADC.B**	Add carry decimally to destination
Syntax	`DADC dst` or `DADC.W dst`
	`DADC.B dst`
Operation	dst + C → dst (decimally)
Emulation	`DADD #0,dst`
	`DADD.B #0,dst`
Description	The carry bit (C) is added decimally to the destination.
Status Bits	N: Set if MSB is 1
	Z: Set if dst is 0, reset otherwise
	C: Set if destination increments from 9999 to 0000, reset otherwise
	Set if destination increments from 99 to 00, reset otherwise
	V: Undefined
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The four-digit decimal number contained in R5 is added to an eight-digit decimal number pointed to by R8.

```
CLRC                    ; Reset carry
                        ; next instruction's start condition is defined
DADD    R5,0(R8)        ; Add LSDs + C
DADC    2(R8)           ; Add carry to MSD
```

Example The two-digit decimal number contained in R5 is added to a four-digit decimal number pointed to by R8.

```
CLRC                    ; Reset carry
                        ; next instruction's start condition is defined
DADD.B    R5,0(R8)      ; Add LSDs + C
DADC      1(R8)         ; Add carry to MSDs
```

4.6.2.16 DADD

*** DADD[.W]**	Add source word and carry decimally to destination word
*** DADD.B**	Add source byte and carry decimally to destination byte

Syntax	`DADD src,dst` or `DADD.W src,dst`
	`DADD.B src,dst`

Operation	src + dst + C → dst (decimally)
Description	The source operand and the destination operand are treated as two (.B) or four (.W) binary coded decimals (BCD) with positive signs. The source operand and the carry bit C are added decimally to the destination operand. The source operand is not affected. The previous content of the destination is lost. The result is not defined for non-BCD numbers.
Status Bits	N: Set if MSB of result is 1 (word > 7999h, byte > 79h), reset if MSB is 0 Z: Set if result is zero, reset otherwise C: Set if the BCD result is too large (word > 9999h, byte > 99h), reset otherwise V: Undefined
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Decimal 10 is added to the 16-bit BCD counter DECCNTR.

```
DADD   #10h,&DECCNTR   ; Add 10 to 4-digit BCD counter
```

Example The eight-digit BCD number contained in 16-bit RAM addresses BCD and BCD+2 is added decimally to an eight-digit BCD number contained in R4 and R5 (BCD+2 and R5 contain the MSDs). The carry C is added, and cleared.

```
CLRC                     ; Clear carry
DADD.W   &BCD,R4         ; Add LSDs. R4.19:16 = 0
DADD.W   &BCD+2,R5       ; Add MSDs with carry. R5.19:16 = 0
JC       OVERFLOW        ; Result >9999,9999: go to error routine
...                      ; Result ok
```

Example The two-digit BCD number contained in word BCD (16-bit address) is added decimally to a two-digit BCD number contained in R4. The carry C is added, also. R4.19:8 = 0

```
CLRC                     ; Clear carry
DADD.B   &BCD,R4         ; Add BCD to R4 decimally.
                           R4: 0,00ddh
```

4.6.2.17 DEC

*** DEC[.W]**	Decrement destination
*** DEC.B**	Decrement destination
Syntax	`DEC dst` or `DEC.W dst`
	`DEC.B dst`
Operation	dst – 1 → dst
Emulation	`SUB #1,dst`
	`SUB.B #1,dst`
Description	The destination operand is decremented by one. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive
	Z: Set if dst contained 1, reset otherwise
	C: Reset if dst contained 0, set otherwise
	V: Set if an arithmetic overflow occurs, otherwise reset.
	Set if initial value of destination was 08000h, otherwise reset.
	Set if initial value of destination was 080h, otherwise reset.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	R10 is decremented by 1.

```
        DEC     R10                 ; Decrement R10

; Move a block of 255 bytes from memory location starting with EDE to
; memory location starting with TONI. Tables should not overlap: start of
; destination address TONI must not be within the range EDE to EDE+0FEh

        MOV     #EDE,R6
        MOV     #255,R10
L$1     MOV.B   @R6+,TONI-EDE-1(R6)
        DEC     R10
        JNZ     L$1
```

Do not transfer tables using the routine above with the overlap shown in Figure 4-36.

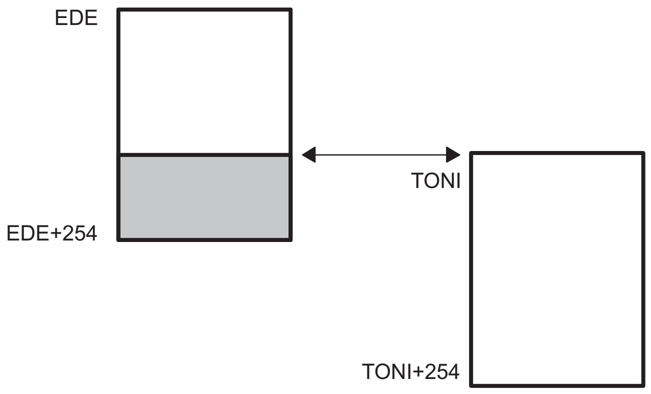

Figure 4-36. Decrement Overlap

4.6.2.18 DECD

*** DECD[.W]**	Double-decrement destination
*** DECD.B**	Double-decrement destination

Syntax	`DECD dst` or `DECD.W dst`
	`DECD.B dst`

Operation	dst – 2 → dst

Emulation	`SUB #2,dst`
	`SUB.B #2,dst`

Description	The destination operand is decremented by two. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive Z: Set if dst contained 2, reset otherwise C: Reset if dst contained 0 or 1, set otherwise V: Set if an arithmetic overflow occurs, otherwise reset Set if initial value of destination was 08001 or 08000h, otherwise reset Set if initial value of destination was 081 or 080h, otherwise reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	R10 is decremented by 2.

```
       DECD     R10             ; Decrement R10 by two

; Move a block of 255 bytes from memory location starting with EDE to
; memory location starting with TONI.
; Tables should not overlap: start of destination address TONI must not
; be within the range EDE to EDE+0FEh

       MOV      #EDE,R6
       MOV      #255,R10
L$1    MOV.B    @R6+,TONI-EDE-2(R6)
       DECD     R10
       JNZ      L$1
```

Example Memory at location LEO is decremented by two.

```
       DECD.B   LEO             ; Decrement MEM(LEO)
```

Decrement status byte STATUS by two

```
       DECD.B   STATUS
```

4.6.2.19 DINT

*** DINT**	Disable (general) interrupts
Syntax	`DINT`
Operation	0 → GIE or (0FFF7h .AND. SR → SR / .NOT.src .AND. dst → dst)
Emulation	`BIC #8,SR`
Description	All interrupts are disabled. The constant 08h is inverted and logically ANDed with the SR. The result is placed into the SR.
Status Bits	Status bits are not affected.
Mode Bits	GIE is reset. OSCOFF and CPUOFF are not affected.
Example	The general interrupt enable (GIE) bit in the SR is cleared to allow a nondisrupted move of a 32-bit counter. This ensures that the counter is not modified during the move by any interrupt.

```
DINT                ; All interrupt events using the GIE bit are disabled
NOP
MOV    COUNTHI,R5   ; Copy counter
MOV    COUNTLO,R6
EINT                ; All interrupt events using the GIE bit are enabled
```

NOTE: Disable interrupt

If any code sequence needs to be protected from interruption, DINT should be executed at least one instruction before the beginning of the uninterruptible sequence, or it should be followed by a NOP instruction.

NOTE: Enable and Disable Interrupt

Due to the pipelined CPU architecture, the instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enabled.

If the enable interrupt instruction (EINT) is immediately followed by a disable interrupt instruction (DINT), a pending interrupt might not be serviced. Further instructions after DINT might execute incorrectly and result in unexpected CPU execution. It is recommended to always insert at least one instruction between EINT and DINT. Note that any alternative instruction use that sets and immediately clears the CPU status register GIE bit must be considered in the same fashion.

4.6.2.20 EINT

*** EINT**	Enable (general) interrupts
Syntax	`EINT`
Operation	1 → GIE or (0008h .OR. SR → SR / .src .OR. dst → dst)
Emulation	`BIS #8,SR`
Description	All interrupts are enabled. The constant #08h and the SR are logically ORed. The result is placed into the SR.
Status Bits	Status bits are not affected.
Mode Bits	GIE is set. OSCOFF and CPUOFF are not affected.
Example	The general interrupt enable (GIE) bit in the SR is set.

```
         PUSH.B   &P1IN
         BIC.B    @SP,&P1IFG  ; Reset only accepted flags
         EINT                 ; Preset port 1 interrupt flags stored on stack
                              ; other interrupts are allowed
         BIT   #Mask,@SP
         JEQ   MaskOK         ; Flags are present identically to mask: jump
         ......
MaskOK   BIC   #Mask,@SP
         ......
         INCD   SP            ; Housekeeping: inverse to PUSH instruction
                              ; at the start of interrupt subroutine. Corrects
                              ; the stack pointer.
         RETI
```

NOTE: Enable and Disable Interrupt

Due to the pipelined CPU architecture, the instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enabled.

If the enable interrupt instruction (EINT) is immediately followed by a disable interrupt instruction (DINT), a pending interrupt might not be serviced. Further instructions after DINT might execute incorrectly and result in unexpected CPU execution. It is recommended to always insert at least one instruction between EINT and DINT. Note that any alternative instruction use that sets and immediately clears the CPU status register GIE bit must be considered in the same fashion.

4.6.2.21 INC

*** INC[.W]**	Increment destination
*** INC.B**	Increment destination
Syntax	`INC dst` or `INC.W dst`
	`INC.B dst`
Operation	dst + 1 → dst
Emulation	`ADD #1,dst`
Description	The destination operand is incremented by one. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive
	Z: Set if dst contained 0FFFFh, reset otherwise
	Set if dst contained 0FFh, reset otherwise
	C: Set if dst contained 0FFFFh, reset otherwise
	Set if dst contained 0FFh, reset otherwise
	V: Set if dst contained 07FFFh, reset otherwise
	Set if dst contained 07Fh, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The status byte, STATUS, of a process is incremented. When it is equal to 11, a branch to OVFL is taken.

```
INC.B   STATUS
CMP.B   #11,STATUS
JEQ     OVFL
```

4.6.2.22 INCD

*** INCD[.W]**	Double-increment destination
*** INCD.B**	Double-increment destination
Syntax	`INCD dst` or `INCD.W dst`
	`INCD.B dst`
Operation	dst + 2 → dst
Emulation	`ADD #2,dst`
Description	The destination operand is incremented by two. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive
	Z: Set if dst contained 0FFFEh, reset otherwise
	Set if dst contained 0FEh, reset otherwise
	C: Set if dst contained 0FFFEh or 0FFFFh, reset otherwise
	Set if dst contained 0FEh or 0FFh, reset otherwise
	V: Set if dst contained 07FFEh or 07FFFh, reset otherwise
	Set if dst contained 07Eh or 07Fh, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The item on the top of the stack (TOS) is removed without using a register.

```
.......
PUSH    R5       ; R5 is the result of a calculation, which is stored
                 ; in the system stack
INCD    SP       ; Remove TOS by double-increment from stack
                 ; Do not use INCD.B, SP is a word-aligned register
RET
```

Example The byte on the top of the stack is incremented by two.

```
INCD.B   0(SP)   ; Byte on TOS is increment by two
```

4.6.2.23 INV

*** INV[.W]**	Invert destination
*** INV.B**	Invert destination

Syntax	`INV dst` or `INV.W dst`
	`INV.B dst`

Operation	.not.dst → dst

Emulation	`XOR #0FFFFh,dst`
	`XOR.B #0FFh,dst`

Description	The destination operand is inverted. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive Z: Set if dst contained 0FFFFh, reset otherwise Set if dst contained 0FFh, reset otherwise C: Set if result is not zero, reset otherwise (= .NOT. Zero) V: Set if initial destination operand was negative, otherwise reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Content of R5 is negated (2s complement).

```
MOV     #00AEh,R5     ;                         R5 = 000AEh
INV     R5            ; Invert R5,              R5 = 0FF51h
INC     R5            ; R5 is now negated,      R5 = 0FF52h
```

Example Content of memory byte LEO is negated.

```
MOV.B   #0AEh,LEO     ;                         MEM(LEO) = 0AEh
INV.B   LEO           ; Invert LEO,             MEM(LEO) = 051h
INC.B   LEO           ; MEM(LEO) is negated,    MEM(LEO) = 052h
```

4.6.2.24 JC, JHS

JC Jump if carry

JHS Jump if higher or same (unsigned)

Syntax `JC label`

`JHS label`

Operation If C = 1: PC + (2 × Offset) → PC
If C = 0: execute the following instruction

Description The carry bit C in the SR is tested. If it is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If C is reset, the instruction after the jump is executed.

JC is used for the test of the carry bit C.

JHS is used for the comparison of unsigned numbers.

Status Bits Status bits are not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The state of the port 1 pin P1IN.1 bit defines the program flow.

```
BIT.B   #2,&P1IN      ; Port 1, bit 1 set? Bit -> C
JC      Label1        ; Yes, proceed at Label1
...                   ; No, continue
```

Example If R5 ≥ R6 (unsigned), the program continues at Label2.

```
CMP     R6,R 5        ; Is R5 >= R6? Info to C
JHS     Label2        ; Yes, C = 1
...                   ; No, R5 < R6. Continue
```

Example If R5 ≥ 12345h (unsigned operands), the program continues at Label2.

```
CMPA    #12345h,R5    ; Is R5 >= 12345h? Info to C
JHS     Label2        ; Yes, 12344h < R5 <= F,FFFFh. C = 1
...                   ; No, R5 < 12345h. Continue
```

4.6.2.25 JEQ, JZ

JEQ	Jump if equal
JZ	Jump if zero

Syntax	`JEQ label`
	`JZ label`

Operation	If Z = 1: PC + (2 × Offset) → PC If Z = 0: execute following instruction
Description	The zero bit Z in the SR is tested. If it is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If Z is reset, the instruction after the jump is executed. JZ is used for the test of the zero bit Z. JEQ is used for the comparison of operands.
Status Bits	Status bits are not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The state of the P2IN.0 bit defines the program flow.

```
BIT.B   #1,&P2IN      ; Port 2, bit 0 reset?
JZ      Label1        ; Yes, proceed at Label1
...                   ; No, set, continue
```

Example If R5 = 15000h (20-bit data), the program continues at Label2.

```
CMPA    #15000h,R5    ; Is R5 = 15000h? Info to SR
JEQ     Label2        ; Yes, R5 = 15000h. Z = 1
...                   ; No, R5 not equal 15000h. Continue
```

Example R7 (20-bit counter) is incremented. If its content is zero, the program continues at Label4.

```
ADDA    #1,R7         ; Increment R7
JZ      Label4        ; Zero reached: Go to Label4
...                   ; R7 not equal 0. Continue here.
```

4.6.2.26 JGE

JGE	Jump if greater or equal (signed)
Syntax	`JGE label`
Operation	If (N .xor. V) = 0: PC + (2 × Offset) → PC If (N .xor. V) = 1: execute following instruction
Description	The negative bit N and the overflow bit V in the SR are tested. If both bits are set or both are reset, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range -511 to +512 words relative to the PC in full Memory range. If only one bit is set, the instruction after the jump is executed. JGE is used for the comparison of signed operands: also for incorrect results due to overflow, the decision made by the JGE instruction is correct. Note that JGE emulates the nonimplemented JP (jump if positive) instruction if used after the instructions AND, BIT, RRA, SXTX, and TST. These instructions clear the V bit.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	If byte EDE (lower 64 K) contains positive data, go to Label1. Software can run in the full memory range.

```
TST.B   &EDE           ; Is EDE positive? V <- 0
JGE     Label1         ; Yes, JGE emulates JP
...                    ; No, 80h <= EDE <= FFh
```

Example If the content of R6 is greater than or equal to the memory pointed to by R7, the program continues a Label5. Signed data. Data and program in full memory range.

```
CMP     @R7,R6         ; Is R6 >= @R7?
JGE     Label5         ; Yes, go to Label5
...                    ; No, continue here
```

Example If R5 ≥ 12345h (signed operands), the program continues at Label2. Program in full memory range.

```
CMPA    #12345h,R5     ; Is R5 >= 12345h?
JGE     Label2         ; Yes, 12344h < R5 <= 7FFFFh
...                    ; No, 80000h <= R5 < 12345h
```

4.6.2.27 JL

JL	Jump if less (signed)
Syntax	`JL label`
Operation	If (N .xor. V) = 1: PC + (2 × Offset) → PC If (N .xor. V) = 0: execute following instruction
Description	The negative bit N and the overflow bit V in the SR are tested. If only one is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in full memory range. If both bits N and V are set or both are reset, the instruction after the jump is executed. JL is used for the comparison of signed operands: also for incorrect results due to overflow, the decision made by the JL instruction is correct.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	If byte EDE contains a smaller, signed operand than byte TONI, continue at Label1. The address EDE is within PC ± 32 K.

```
CMP.B   &TONI,EDE      ; Is EDE < TONI
JL      Label1         ; Yes
...                    ; No, TONI <= EDE
```

Example If the signed content of R6 is less than the memory pointed to by R7 (20-bit address), the program continues at Label5. Data and program in full memory range.

```
CMP     @R7,R6         ; Is R6 < @R7?
JL      Label5         ; Yes, go to Label5
...                    ; No, continue here
```

Example If R5 < 12345h (signed operands), the program continues at Label2. Data and program in full memory range.

```
CMPA    #12345h,R5     ; Is R5 < 12345h?
JL      Label2         ; Yes, 80000h =< R5 < 12345h
...                    ; No, 12344h < R5 <= 7FFFFh
```

4.6.2.28 JMP

JMP	Jump unconditionally
Syntax	`JMP label`
Operation	PC + (2 × Offset) → PC
Description	The signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means an unconditional jump in the range –511 to +512 words relative to the PC in the full memory. The JMP instruction may be used as a BR or BRA instruction within its limited range relative to the PC.
Status Bits	Status bits are not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The byte STATUS is set to 10. Then a jump to label MAINLOOP is made. Data in lower 64 K, program in full memory range.

```
MOV.B   #10,&STATUS     ; Set STATUS to 10
JMP     MAINLOOP        ; Go to main loop
```

Example The interrupt vector TAIV of Timer_A3 is read and used for the program flow. Program in full memory range, but interrupt handlers always starts in lower 64 K.

```
ADD     &TAIV,PC        ; Add Timer_A interrupt vector to PC
RETI                    ; No Timer_A interrupt pending
JMP     IHCCR1          ; Timer block 1 caused interrupt
JMP     IHCCR2          ; Timer block 2 caused interrupt
RETI                    ; No legal interrupt, return
```

4.6.2.29 JN

JN	Jump if negative
Syntax	`JN label`
Operation	If N = 1: PC + (2 × Offset) → PC If N = 0: execute following instruction
Description	The negative bit N in the SR is tested. If it is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program PC. This means a jump in the range -511 to +512 words relative to the PC in the full memory range. If N is reset, the instruction after the jump is executed.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The byte COUNT is tested. If it is negative, program execution continues at Label0. Data in lower 64 K, program in full memory range.

```
TST.B   &COUNT      ; Is byte COUNT negative?
JN      Label0      ; Yes, proceed at Label0
...                 ; COUNT >= 0
```

Example R6 is subtracted from R5. If the result is negative, program continues at Label2. Program in full memory range.

```
SUB     R6,R5       ; R5 - R6 -> R5
JN      Label2      ; R5 is negative: R6 > R5 (N = 1)
...                 ; R5 >= 0. Continue here.
```

Example R7 (20-bit counter) is decremented. If its content is below zero, the program continues at Label4. Program in full memory range.

```
SUBA    #1,R7       ; Decrement R7
JN      Label4      ; R7 < 0: Go to Label4
...                 ; R7 >= 0. Continue here.
```

4.6.2.30 JNC, JLO

JNC	Jump if no carry
JLO	Jump if lower (unsigned)

Syntax	`JNC label`
	 `JLO label`

Operation	If C = 0: PC + (2 × Offset) → PC If C = 1: execute following instruction
Description	The carry bit C in the SR is tested. If it is reset, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If C is set, the instruction after the jump is executed. JNC is used for the test of the carry bit C. JLO is used for the comparison of unsigned numbers.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	If byte EDE < 15, the program continues at Label2. Unsigned data. Data in lower 64 K, program in full memory range.

```
CMP.B   #15,&EDE      ; Is EDE < 15? Info to C
JLO     Label2        ; Yes, EDE < 15. C = 0
...                   ; No, EDE >= 15. Continue
```

Example The word TONI is added to R5. If no carry occurs, continue at Label0. The address of TONI is within PC ± 32 K.

```
ADD     TONI,R5       ; TONI + R5 -> R5. Carry -> C
JNC     Label0        ; No carry
...                   ; Carry = 1: continue here
```

4.6.2.31 JNZ, JNE

JNZ	Jump if not zero
JNE	Jump if not equal

Syntax	`JNZ label`
	`JNE label`

Operation	If Z = 0: PC + (2 × Offset) → PC If Z = 1: execute following instruction
Description	The zero bit Z in the SR is tested. If it is reset, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If Z is set, the instruction after the jump is executed. JNZ is used for the test of the zero bit Z. JNE is used for the comparison of operands.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The byte STATUS is tested. If it is not zero, the program continues at Label3. The address of STATUS is within PC ± 32 K.

```
TST.B   STATUS          ; Is STATUS = 0?
JNZ     Label3          ; No, proceed at Label3
...                     ; Yes, continue here
```

Example If word EDE ≠ 1500, the program continues at Label2. Data in lower 64 K, program in full memory range.

```
CMP     #1500,&EDE      ; Is EDE = 1500? Info to SR
JNE     Label2          ; No, EDE not equal 1500.
...                     ; Yes, R5 = 1500. Continue
```

Example R7 (20-bit counter) is decremented. If its content is not zero, the program continues at Label4. Program in full memory range.

```
SUBA     #1,R7          ; Decrement R7
JNZ      Label4         ; Zero not reached: Go to Label4
...                     ; Yes, R7 = 0. Continue here.
```

4.6.2.32 MOV

MOV[.W]	Move source word to destination word
MOV.B	Move source byte to destination byte
Syntax	`MOV src,dst` or `MOV.W src,dst`
	`MOV.B src,dst`
Operation	src → dst
Description	The source operand is copied to the destination. The source operand is not affected.
Status Bits	N: Not affected
	Z: Not affected
	C: Not affected
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Move a 16-bit constant 1800h to absolute address-word EDE (lower 64 K)

```
MOV     #01800h,&EDE            ; Move 1800h to EDE
```

Example The contents of table EDE (word data, 16-bit addresses) are copied to table TOM. The length of the tables is 030h words. Both tables reside in the lower 64 K.

```
        MOV     #EDE,R10                ; Prepare pointer (16-bit address)
Loop    MOV     @R10+,TOM-EDE-2(R10)    ; R10 points to both tables.
                                        ; R10+2
        CMP     #EDE+60h,R10            ; End of table reached?
        JLO     Loop                    ; Not yet
        ...                             ; Copy completed
```

Example The contents of table EDE (byte data, 16-bit addresses) are copied to table TOM. The length of the tables is 020h bytes. Both tables may reside in full memory range, but must be within R10 ± 32 K.

```
        MOVA    #EDE,R10                ; Prepare pointer (20-bit)
        MOV     #20h,R9                 ; Prepare counter
Loop    MOV.B   @R10+,TOM-EDE-1(R10)    ; R10 points to both tables.
                                        ; R10+1
        DEC     R9                      ; Decrement counter
        JNZ     Loop                    ; Not yet done
        ...                             ; Copy completed
```

4.6.2.33 NOP

*** NOP**	No operation
Syntax	`NOP`
Operation	None
Emulation	`MOV #0, R3`
Description	No operation is performed. The instruction may be used for the elimination of instructions during the software check or for defined waiting times.
Status Bits	Status bits are not affected.

4.6.2.34 POP

*** POP[.W]**	Pop word from stack to destination
*** POP.B**	Pop byte from stack to destination

Syntax	`POP dst`
	 `POP.B dst`

Operation	@SP → temp SP + 2 → SP temp → dst

Emulation	`MOV @SP+,dst` or `MOV.W @SP+,dst`
	 `MOV.B @SP+,dst`

Description	The stack location pointed to by the SP (TOS) is moved to the destination. The SP is incremented by two afterwards.
Status Bits	Status bits are not affected.
Example	The contents of R7 and the SR are restored from the stack.

```
POP     R7      ; Restore R7
POP     SR      ; Restore status register
```

Example The contents of RAM byte LEO is restored from the stack.

```
POP.B   LEO     ; The low byte of the stack is moved to LEO.
```

Example The contents of R7 is restored from the stack.

```
POP.B   R7      ; The low byte of the stack is moved to R7,
                ; the high byte of R7 is 00h
```

Example The contents of the memory pointed to by R7 and the SR are restored from the stack.

```
POP.B   0(R7)   ; The low byte of the stack is moved to the
                ; the byte which is pointed to by R7
                : Example:   R7 = 203h
                ;            Mem(R7) = low byte of system stack
                : Example:   R7 = 20Ah
                ;            Mem(R7) = low byte of system stack
POP     SR      ; Last word on stack moved to the SR
```

NOTE: System stack pointer

The system SP is always incremented by two, independent of the byte suffix.

4.6.2.35 PUSH

PUSH[.W]	Save a word on the stack
PUSH.B	Save a byte on the stack

Syntax	`PUSH dst` or `PUSH.W dst`
	`PUSH.B dst`

Operation	SP – 2 → SP dst → @SP
Description	The 20-bit SP SP is decremented by two. The operand is then copied to the RAM word addressed by the SP. A pushed byte is stored in the low byte; the high byte is not affected.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Save the two 16-bit registers R9 and R10 on the stack

```
PUSH     R9       ; Save R9 and R10 XXXXh
PUSH     R10      ; YYYYh
```

Example Save the two bytes EDE and TONI on the stack. The addresses EDE and TONI are within PC ± 32 K.

```
PUSH.B   EDE      ; Save EDE   xxXXh
PUSH.B   TONI     ; Save TONI   xxYYh
```

4.6.2.36 RET

*** RET**	Return from subroutine
Syntax	`RET`
Operation	@SP →PC.15:0 Saved PC to PC.15:0. PC.19:16 ← 0 SP + 2 → SP
Description	The 16-bit return address (lower 64 K), pushed onto the stack by a CALL instruction is restored to the PC. The program continues at the address following the subroutine call. The four MSBs of the PC.19:16 are cleared.
Status Bits	Status bits are not affected. PC.19:16: Cleared
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Call a subroutine SUBR in the lower 64 K and return to the address in the lower 64 K after the CALL.

```
        CALL    #SUBR      ; Call subroutine starting at SUBR
        ...                ; Return by RET to here
SUBR    PUSH    R14        ; Save R14 (16 bit data)
        ...                ; Subroutine code
        POP     R14        ; Restore R14
        RET                ; Return to lower 64 K
```

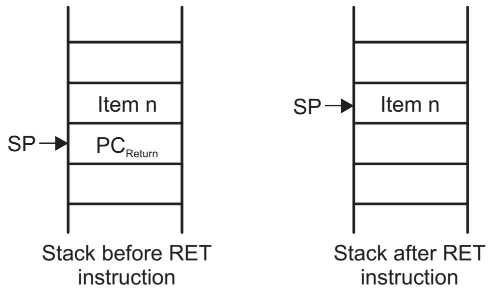

Figure 4-37. Stack After a RET Instruction

4.6.2.37 RETI

RETI	Return from interrupt
Syntax	`RETI`
Operation	@SP → SR.15:0 Restore saved SR with PC.19:16 SP + 2 → SP @SP → PC.15:0 Restore saved PC.15:0 SP + 2 → SP Housekeeping
Description	The SR is restored to the value at the beginning of the interrupt service routine. This includes the four MSBs of the PC.19:16. The SP is incremented by two afterward. The 20-bit PC is restored from PC.19:16 (from same stack location as the status bits) and PC.15:0. The 20-bit PC is restored to the value at the beginning of the interrupt service routine. The program continues at the address following the last executed instruction when the interrupt was granted. The SP is incremented by two afterward.
Status Bits	N: Restored from stack C: Restored from stack Z: Restored from stack V: Restored from stack
Mode Bits	OSCOFF, CPUOFF, and GIE are restored from stack.
Example	Interrupt handler in the lower 64 K. A 20-bit return address is stored on the stack.

```
INTRPT  PUSHM.A   #2,R14    ; Save R14 and R13 (20-bit data)
        ...                 ; Interrupt handler code
        POPM.A    #2,R14    ; Restore R13 and R14 (20-bit data)
        RETI                ; Return to 20-bit address in full memory range
```

4.6.2.38 RLA

*** RLA[.W]**	Rotate left arithmetically
*** RLA.B**	Rotate left arithmetically

Syntax	`RLA dst` or `RLA.W dst`
	`RLA.B dst`

Operation	C ← MSB ← MSB-1 .... LSB+1 ← LSB ← 0

Emulation	`ADD dst,dst`
	`ADD.B dst,dst`

Description The destination operand is shifted left one position as shown in Figure 4-38. The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLA instruction acts as a signed multiplication by 2.

An overflow occurs if dst ≥ 04000h and dst < 0C000h before operation is performed; the result has changed sign.

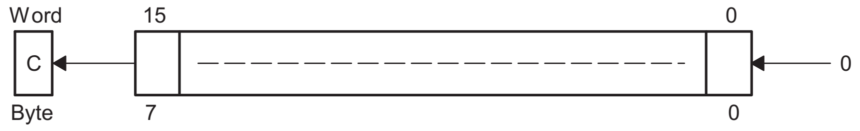

Figure 4-38. Destination Operand—Arithmetic Shift Left

An overflow occurs if dst ≥ 040h and dst < 0C0h before the operation is performed; the result has changed sign.

Status Bits
- N: Set if result is negative, reset if positive
- Z: Set if result is zero, reset otherwise
- C: Loaded from the MSB
- V: Set if an arithmetic overflow occurs; the initial value is 04000h ≤ dst < 0C000h, reset otherwise

  Set if an arithmetic overflow occurs; the initial value is 040h ≤ dst < 0C0h, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example R7 is multiplied by 2.

```
RLA      R7    ; Shift left R7  (x 2)
```

Example The low byte of R7 is multiplied by 4.

```
RLA.B    R7    ; Shift left low byte of R7  (x 2)
RLA.B    R7    ; Shift left low byte of R7  (x 4)
```

NOTE: RLA substitution

The assembler does not recognize the instructions:

```
RLA  @R5+          RLA.B  @R5+          RLA(.B) @R5
```

They must be substituted by:

```
ADD  @R5+,-2(R5)   ADD.B  @R5+,-1(R5)   ADD(.B) @R5
```

4.6.2.39 RLC

*** RLC[.W]**	Rotate left through carry
*** RLC.B**	Rotate left through carry

Syntax	`RLC dst` or `RLC.W dst`
	 `RLC.B dst`

Operation	C ← MSB ← MSB-1 .... LSB+1 ← LSB ← C
Emulation	`ADDC dst,dst`
Description	The destination operand is shifted left one position as shown in Figure 4-39. The carry bit (C) is shifted into the LSB, and the MSB is shifted into the carry bit (C).

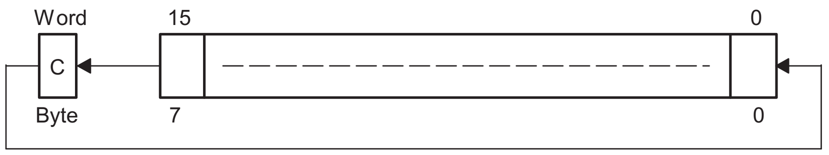

Figure 4-39. Destination Operand—Carry Left Shift

Status Bits

N: Set if result is negative, reset if positive

Z: Set if result is zero, reset otherwise

C: Loaded from the MSB

V: Set if an arithmetic overflow occurs; the initial value is 04000h ≤ dst < 0C000h, reset otherwise

Set if an arithmetic overflow occurs; the initial value is 040h ≤ dst < 0C0h, reset otherwise

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example R5 is shifted left one position.

```
    RLC      R5            ; (R5 x 2) + C -> R5
```

Example The input P1IN.1 information is shifted into the LSB of R5.

```
    BIT.B    #2,&P1IN      ; Information -> Carry
    RLC      R5            ; Carry=P0in.1 -> LSB of R5
```

Example The MEM(LEO) content is shifted left one position.

```
    RLC.B    LEO           ; Mem(LEO) x 2 + C -> Mem(LEO)
```

NOTE: RLA substitution

The assembler does not recognize the instructions:

```
RLC   @R5+           RLC.B   @R5+           RLC(.B) @R5
```

They must be substituted by:

```
ADDC  @R5+,-2(R5)    ADDC.B  @R5+,-1(R5)    ADDC(.B) @R5
```

4.6.2.40 RRA

RRA[.W]	Rotate right arithmetically destination word
RRA.B	Rotate right arithmetically destination byte
Syntax	`RRA.B dst` or `RRA.W dst`
Operation	MSB → MSB → MSB–1 → ... LSB+1 → LSB → C
Description	The destination operand is shifted right arithmetically by one bit position as shown in Figure 4-40. The MSB retains its value (sign). RRA operates equal to a signed division by 2. The MSB is retained and shifted into the MSB–1. The LSB+1 is shifted into the LSB. The previous LSB is shifted into the carry bit C.
Status Bits	N: Set if result is negative (MSB = 1), reset otherwise (MSB = 0)
	Z: Set if result is zero, reset otherwise
	C: Loaded from the LSB
	V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The signed 16-bit number in R5 is shifted arithmetically right one position.

```
RRA     R5              ; R5/2 -> R5
```

Example The signed RAM byte EDE is shifted arithmetically right one position.

```
RRA.B   EDE             ; EDE/2 -> EDE
```

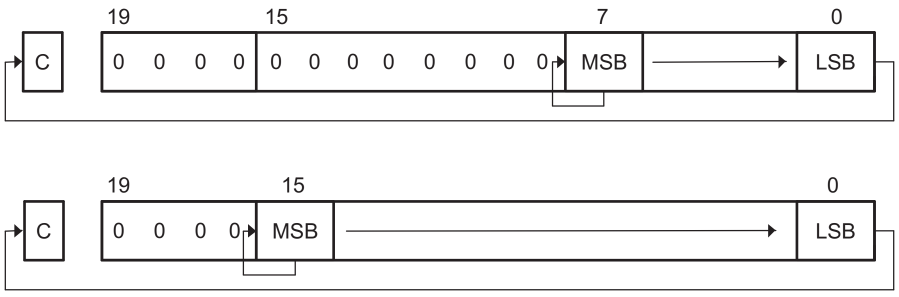

Figure 4-40. Rotate Right Arithmetically RRA.B and RRA.W

4.6.2.41 RRC

RRC[.W]	Rotate right through carry destination word
RRC.B	Rotate right through carry destination byte
Syntax	`RRC dst` or `RRC.W dst`
	`RRC.B dst`
Operation	C → MSB → MSB–1 → ... LSB+1 → LSB → C
Description	The destination operand is shifted right by one bit position as shown in Figure 4-41. The carry bit C is shifted into the MSB and the LSB is shifted into the carry bit C.
Status Bits	N: Set if result is negative (MSB = 1), reset otherwise (MSB = 0)
	Z: Set if result is zero, reset otherwise
	C: Loaded from the LSB
	V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	RAM word EDE is shifted right one bit position. The MSB is loaded with 1.

```
SETC            ; Prepare carry for MSB
RRC    EDE      ; EDE = EDE >> 1 + 8000h
```

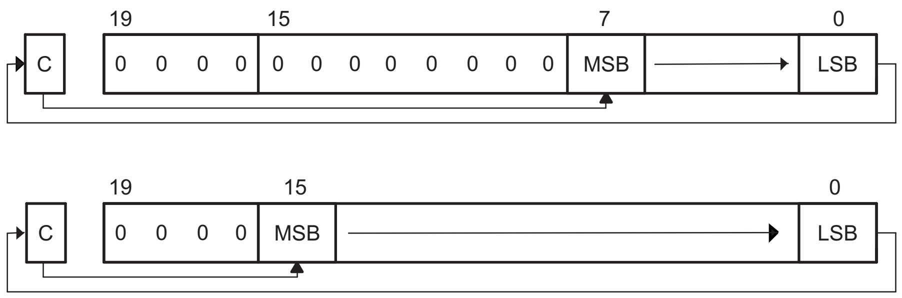

Figure 4-41. Rotate Right Through Carry RRC.B and RRC.W

4.6.2.42 SBC

*** SBC[.W]**	Subtract borrow (.NOT. carry) from destination
*** SBC.B**	Subtract borrow (.NOT. carry) from destination

Syntax	`SBC dst` or `SBC.W dst`
	`SBC.B dst`

Operation	dst + 0FFFFh + C → dst dst + 0FFh + C → dst

Emulation	`SUBC #0,dst`
	`SUBC.B #0,dst`

Description	The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.
Status Bits	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB of the result, reset otherwise Set to 1 if no borrow, reset if borrow V: Set if an arithmetic overflow occurs, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 16-bit counter pointed to by R13 is subtracted from a 32-bit counter pointed to by R12.

```
SUB     @R13,0(R12)     ; Subtract LSDs
SBC     2(R12)          ; Subtract carry from MSD
```

Example The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.

```
SUB.B   @R13,0(R12)     ; Subtract LSDs
SBC.B   1(R12)          ; Subtract carry from MSD
```

NOTE: Borrow implementation

The borrow is treated as a .NOT. carry:

Borrow	Carry Bit
Yes	0
No	1

4.6.2.43 SETC

*** SETC**	Set carry bit
Syntax	`SETC`
Operation	1 → C
Emulation	`BIS #1,SR`
Description	The carry bit (C) is set.
Status Bits	N: Not affected
	Z: Not affected
	C: Set
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Emulation of the decimal subtraction: Subtract R5 from R6 decimally. Assume that R5 = 03987h and R6 = 04137h.

```
DSUB   ADD    #06666h,R5     ; Move content R5 from 0-9 to 6-0Fh
                             ; R5 = 03987h + 06666h = 09FEDh
       INV    R5             ; Invert this (result back to 0-9)
                             ; R5 = .NOT. R5 = 06012h
       SETC                  ; Prepare carry = 1
       DADD   R5,R6          ; Emulate subtraction by addition of:
                             ; (010000h - R5 - 1)
                             ; R6 = R6 + R5 + 1
                             ; R6 = 0150h
```

4.6.2.44 SETN

*** SETN**	Set negative bit
Syntax	`SETN`
Operation	1 → N
Emulation	`BIS #4,SR`
Description	The negative bit (N) is set.
Status Bits	N: Set
	Z: Not affected
	C: Not affected
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.

4.6.2.45 SETZ

*** SETZ**	Set zero bit
Syntax	`SETZ`
Operation	1 → N
Emulation	`BIS #2,SR`
Description	The zero bit (Z) is set.
Status Bits	N: Not affected
	Z: Set
	C: Not affected
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.

4.6.2.46 SUB

SUB[.W]	Subtract source word from destination word
SUB.B	Subtract source byte from destination byte
Syntax	`SUB src,dst` or `SUB.W src,dst`
	`SUB.B src,dst`
Operation	(.not.src) + 1 + dst → dst or dst – src → dst
Description	The source operand is subtracted from the destination operand. This is made by adding the 1s complement of the source + 1 to the destination. The source operand is not affected, the result is written to the destination operand.
Status Bits	N: Set if result is negative (src > dst), reset if positive (src ≤ dst)
	Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)
	C: Set if there is a carry from the MSB, reset otherwise
	V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	A 16-bit constant 7654h is subtracted from RAM word EDE.

```
SUB     #7654h,&EDE      ; Subtract 7654h from EDE
```

Example A table word pointed to by R5 (20-bit address) is subtracted from R7. Afterwards, if R7 contains zero, jump to label TONI. R5 is then auto-incremented by 2. R7.19:16 = 0.

```
SUB     @R5+,R7         ; Subtract table number from R7. R5 + 2
JZ      TONI            ; R7 = @R5 (before subtraction)
...                     ; R7 <> @R5 (before subtraction)
```

Example Byte CNT is subtracted from byte R12 points to. The address of CNT is within PC ± 32K. The address R12 points to is in full memory range.

```
SUB.B   CNT,0(R12)      ; Subtract CNT from @R12
```

4.6.2.47 SUBC

SUBC[.W]	Subtract source word with carry from destination word
SUBC.B	Subtract source byte with carry from destination byte

Syntax	`SUBC src,dst` or `SUBC.W src,dst`
	`SUBC.B src,dst`

Operation	(.not.src) + C + dst → dst or dst – (src – 1) + C → dst
Description	The source operand is subtracted from the destination operand. This is done by adding the 1s complement of the source + carry to the destination. The source operand is not affected, the result is written to the destination operand. Used for 32, 48, and 64-bit operands.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB, reset otherwise V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	A 16-bit constant 7654h is subtracted from R5 with the carry from the previous instruction. R5.19:16 = 0

```
SUBC.W   #7654h,R5       ; Subtract 7654h + C from R5
```

Example A 48-bit number (3 words) pointed to by R5 (20-bit address) is subtracted from a 48-bit counter in RAM, pointed to by R7. R5 points to the next 48-bit number afterwards. The address R7 points to is in full memory range.

```
SUB      @R5+,0(R7)      ; Subtract LSBs. R5 + 2
SUBC     @R5+,2(R7)      ; Subtract MIDs with C. R5 + 2
SUBC     @R5+,4(R7)      ; Subtract MSBs with C. R5 + 2
```

Example Byte CNT is subtracted from the byte, R12 points to. The carry of the previous instruction is used. The address of CNT is in lower 64 K.

```
SUBC.B   &CNT,0(R12)     ; Subtract byte CNT from @R12
```

4.6.2.48 SWPB

SWPB	Swap bytes
Syntax	`SWPB dst`
Operation	dst.15:8 ↔ dst.7:0
Description	The high and the low byte of the operand are exchanged. PC.19:16 bits are cleared in register mode.
Status Bits	Status bits are not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Exchange the bytes of RAM word EDE (lower 64 K)

```
MOV     #1234h,&EDE       ; 1234h -> EDE
SWPB    &EDE              ; 3412h -> EDE
```

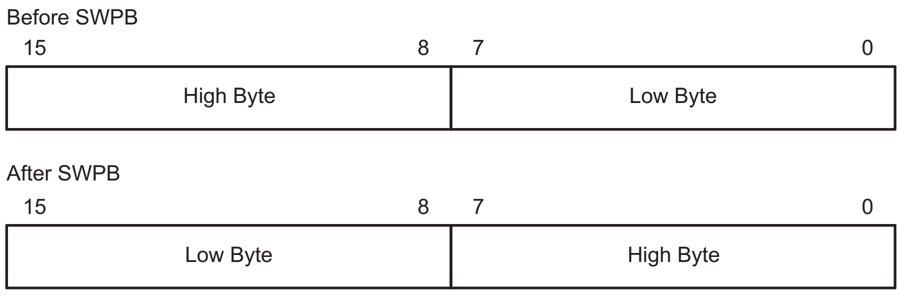

Figure 4-42. Swap Bytes in Memory

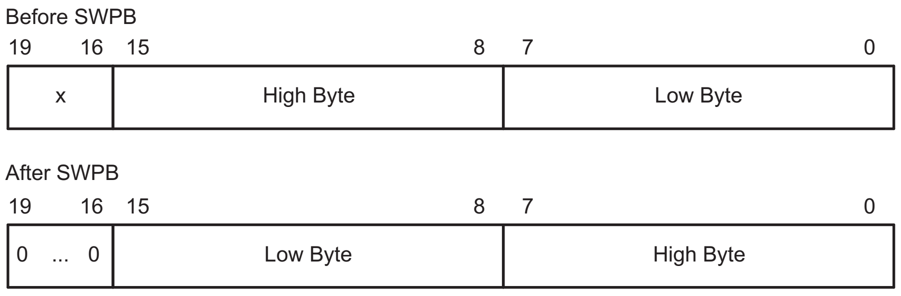

Figure 4-43. Swap Bytes in a Register

4.6.2.49 SXT

SXT	Extend sign
Syntax	`SXT dst`
Operation	dst.7 → dst.15:8, dst.7 → dst.19:8 (register mode)

Description	Register mode: the sign of the low byte of the operand is extended into the bits Rdst.19:8.
	Rdst.7 = 0: Rdst.19:8 = 000h afterwards
	Rdst.7 = 1: Rdst.19:8 = FFFh afterwards
	Other modes: the sign of the low byte of the operand is extended into the high byte.
	dst.7 = 0: high byte = 00h afterwards
	dst.7 = 1: high byte = FFh afterwards

Status Bits	N: Set if result is negative, reset otherwise Z: Set if result is zero, reset otherwise C: Set if result is not zero, reset otherwise (C = .not.Z) V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The signed 8-bit data in EDE (lower 64 K) is sign extended and added to the 16-bit signed data in R7.

```
MOV.B   &EDE,R5      ; EDE -> R5. 00XXh
SXT     R5           ; Sign extend low byte to R5.19:8
ADD     R5,R7        ; Add signed 16-bit values
```

Example The signed 8-bit data in EDE (PC +32 K) is sign extended and added to the 20-bit data in R7.

```
MOV.B   EDE,R5       ; EDE -> R5. 00XXh
SXT     R5           ; Sign extend low byte to R5.19:8
ADDA    R5,R7        ; Add signed 20-bit values
```

4.6.2.50 TST

*** TST[.W]**	Test destination
*** TST.B**	Test destination

Syntax	`TST dst` or `TST.W dst`
	`TST.B dst`

Operation	dst + 0FFFFh + 1 dst + 0FFh + 1

Emulation	`CMP #0,dst`
	`CMP.B #0,dst`

Description	The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.
Status Bits	N: Set if destination is negative, reset if positive Z: Set if destination contains zero, reset otherwise C: Set V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
         TST     R7        ; Test R7
         JN      R7NEG     ; R7 is negative
         JZ      R7ZERO    ; R7 is zero
R7POS    ......            ; R7 is positive but not zero
R7NEG    ......            ; R7 is negative
R7ZERO   ......            ; R7 is zero
```

Example The low byte of R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
         TST.B   R7        ; Test low byte of R7
         JN      R7NEG     ; Low byte of R7 is negative
         JZ      R7ZERO    ; Low byte of R7 is zero
R7POS    ......            ; Low byte of R7 is positive but not zero
R7NEG    .....             ; Low byte of R7 is negative
R7ZERO   ......            ; Low byte of R7 is zero
```

4.6.2.51 XOR

XOR[.W]	Exclusive OR source word with destination word
XOR.B	Exclusive OR source byte with destination byte

Syntax	`XOR src,dst` or `XOR.W src,dst`
	`XOR.B src,dst`

Operation	src .xor. dst → dst
Description	The source and destination operands are exclusively ORed. The result is placed into the destination. The source operand is not affected. The previous content of the destination is lost.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if result is not zero, reset otherwise (C = .not. Z) V: Set if both operands are negative before execution, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Toggle bits in word CNTR (16-bit data) with information (bit = 1) in address-word TONI. Both operands are located in lower 64 K.

```
XOR     &TONI,&CNTR     ; Toggle bits in CNTR
```

Example A table word pointed to by R5 (20-bit address) is used to toggle bits in R6. R6.19:16 = 0.

```
XOR     @R5,R6          ; Toggle bits in R6
```

Example Reset to zero those bits in the low byte of R7 that are different from the bits in byte EDE. R7.19:8 = 0. The address of EDE is within PC ± 32 K.

```
XOR.B   EDE,R7          ; Set different bits to 1 in R7.
INV.B   R7              ; Invert low byte of R7, high byte is 0h
```

4.6.3 Extended Instructions

The extended MSP430X instructions give the MSP430X CPU full access to its 20-bit address space. MSP430X instructions require an additional word of op-code called the extension word. All addresses, indexes, and immediate numbers have 20-bit values when preceded by the extension word. The MSP430X extended instructions are listed and described in the following pages.

4.6.3.1 ADCX

*** ADCX.A**	Add carry to destination address-word
*** ADCX.[W]**	Add carry to destination word
*** ADCX.B**	Add carry to destination byte

Syntax	`ADCX.A dst`
	`ADCX dst` or `ADCX.W dst`
	`ADCX.B dst`

Operation	dst + C → dst

Emulation	`ADDCX.A #0,dst`
	`ADDCX #0,dst`
	`ADDCX.B #0,dst`

Description	The carry bit (C) is added to the destination operand. The previous contents of the destination are lost.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB of the result, reset otherwise V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 40-bit counter, pointed to by R12 and R13, is incremented.

```
INCX.A   @R12     ; Increment lower 20 bits
ADCX.A   @R13     ; Add carry to upper 20 bits
```

4.6.3.2 ADDX

ADDX.A	Add source address-word to destination address-word
ADDX.[W]	Add source word to destination word
ADDX.B	Add source byte to destination byte

Syntax	`ADDX.A src,dst`
	`ADDX src,dst` or `ADDX.W src,dst`
	`ADDX.B src,dst`

Operation	src + dst → dst
Description	The source operand is added to the destination operand. The previous contents of the destination are lost. Both operands can be located in the full address space.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB of the result, reset otherwise V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Ten is added to the 20-bit pointer CNTR located in two words CNTR (LSBs) and CNTR+2 (MSBs).

```
ADDX.A   #10,CNTR    ; Add 10 to 20-bit pointer
```

Example A table word (16-bit) pointed to by R5 (20-bit address) is added to R6. The jump to label TONI is performed on a carry.

```
ADDX.W   @R5,R6      ; Add table word to R6
JC       TONI        ; Jump if carry
...                  ; No carry
```

Example A table byte pointed to by R5 (20-bit address) is added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1.

```
ADDX.B   @R5+,R6     ; Add table byte to R6. R5 + 1. R6: 000xxh
JNC      TONI        ; Jump if no carry
...                  ; Carry occurred
```

Note: Use ADDA for the following two cases for better code density and execution.

```
ADDX.A   Rsrc,Rdst
ADDX.A   #imm20,Rdst
```

4.6.3.3 ADDCX

ADDCX.A	Add source address-word and carry to destination address-word
ADDCX.[W]	Add source word and carry to destination word
ADDCX.B	Add source byte and carry to destination byte

Syntax	`ADDCX.A src,dst`
	`ADDCX src,dst` or `ADDCX.W src,dst`
	`ADDCX.B src,dst`

Operation	src + dst + C → dst
Description	The source operand and the carry bit C are added to the destination operand. The previous contents of the destination are lost. Both operands may be located in the full address space.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB of the result, reset otherwise V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Constant 15 and the carry of the previous instruction are added to the 20-bit counter CNTR located in two words.

```
ADDCX.A   #15,&CNTR   ; Add 15 + C to 20-bit CNTR
```

Example A table word pointed to by R5 (20-bit address) and the carry C are added to R6. The jump to label TONI is performed on a carry.

```
ADDCX.W   @R5,R6      ; Add table word + C to R6
JC        TONI        ; Jump if carry
...                   ; No carry
```

Example A table byte pointed to by R5 (20-bit address) and the carry bit C are added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1.

```
ADDCX.B   @R5+,R6     ; Add table byte + C to R6. R5 + 1
JNC       TONI        ; Jump if no carry
...                   ; Carry occurred
```

4.6.3.4 ANDX

ANDX.A	Logical AND of source address-word with destination address-word
ANDX.[W]	Logical AND of source word with destination word
ANDX.B	Logical AND of source byte with destination byte

Syntax	`ANDX.A src,dst`
	`ANDX src,dst` or `ANDX.W src,dst`
	`ANDX.B src,dst`

Operation	src .and. dst → dst
Description	The source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected. Both operands may be located in the full address space.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if the result is not zero, reset otherwise. C = (.not. Z) V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The bits set in R5 (20-bit data) are used as a mask (AAA55h) for the address-word TOM located in two words. If the result is zero, a branch is taken to label TONI.

```
MOVA     #AAA55h,R5      ; Load 20-bit mask to R5
ANDX.A   R5,TOM          ; TOM .and. R5 -> TOM
JZ       TONI            ; Jump if result 0
...                      ; Result > 0
```

or shorter:

```
ANDX.A   #AAA55h,TOM     ; TOM .and. AAA55h -> TOM
JZ       TONI            ; Jump if result 0
```

Example A table byte pointed to by R5 (20-bit address) is logically ANDed with R6. R6.19:8 = 0. The table pointer is auto-incremented by 1.

```
ANDX.B   @R5+,R6         ; AND table byte with R6. R5 + 1
```

4.6.3.5 BICX

BICX.A	Clear bits set in source address-word in destination address-word
BICX.[W]	Clear bits set in source word in destination word
BICX.B	Clear bits set in source byte in destination byte

Syntax	`BICX.A src,dst`
	`BICX src,dst` or `BICX.W src,dst`
	`BICX.B src,dst`

Operation	(.not. src) .and. dst → dst
Description	The inverted source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected. Both operands may be located in the full address space.
Status Bits	N: Not affected Z: Not affected C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The bits 19:15 of R5 (20-bit data) are cleared.

```
BICX.A   #0F8000h,R5     ; Clear R5.19:15 bits
```

Example A table word pointed to by R5 (20-bit address) is used to clear bits in R7. R7.19:16 = 0.

```
BICX.W   @R5,R7          ; Clear bits in R7
```

Example A table byte pointed to by R5 (20-bit address) is used to clear bits in output Port1.

```
BICX.B   @R5,&P1OUT      ; Clear I/O port P1 bits
```

4.6.3.6 BISX

BISX.A	Set bits set in source address-word in destination address-word
BISX.[W]	Set bits set in source word in destination word
BISX.B	Set bits set in source byte in destination byte

Syntax	`BISX.A src,dst`
	`BISX src,dst` or `BISX.W src,dst`
	`BISX.B src,dst`

Operation	src .or. dst → dst
Description	The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected. Both operands may be located in the full address space.
Status Bits	N: Not affected Z: Not affected C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Bits 16 and 15 of R5 (20-bit data) are set to one.

```
BISX.A    #018000h,R5     ; Set R5.16:15 bits
```

Example A table word pointed to by R5 (20-bit address) is used to set bits in R7.

```
BISX.W   @R5,R7           ; Set bits in R7
```

Example A table byte pointed to by R5 (20-bit address) is used to set bits in output Port1.

```
BISX.B   @R5,&P1OUT       ; Set I/O port P1 bits
```

4.6.3.7 BITX

BITX.A	Test bits set in source address-word in destination address-word
BITX.[W]	Test bits set in source word in destination word
BITX.B	Test bits set in source byte in destination byte

Syntax	`BITX.A src,dst`
	`BITX src,dst` or `BITX.W src,dst`
	`BITX.B src,dst`

Operation	src .and. dst → dst
Description	The source operand and the destination operand are logically ANDed. The result affects only the status bits. Both operands may be located in the full address space.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if the result is not zero, reset otherwise. C = (.not. Z) V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Test if bit 16 or 15 of R5 (20-bit data) is set. Jump to label TONI if so.

```
BITX.A   #018000h,R5      ; Test R5.16:15 bits
JNZ      TONI             ; At least one bit is set
...                       ; Both are reset
```

Example A table word pointed to by R5 (20-bit address) is used to test bits in R7. Jump to label TONI if at least one bit is set.

```
BITX.W   @R5,R7            ; Test bits in R7: C = .not.Z
JC       TONI              ; At least one is set
...                        ; Both are reset
```

Example A table byte pointed to by R5 (20-bit address) is used to test bits in input Port1. Jump to label TONI if no bit is set. The next table byte is addressed.

```
BITX.B   @R5+,&P1IN        ; Test input P1 bits. R5 + 1
JNC      TONI              ; No corresponding input bit is set
...                        ; At least one bit is set
```

4.6.3.8 CLRX

*** CLRX.A**	Clear destination address-word
*** CLRX.[W]**	Clear destination word
*** CLRX.B**	Clear destination byte

Syntax	`CLRX.A dst`
	`CLRX dst` or `CLRX.W dst`
	`CLRX.B dst`

Operation	0 → dst

Emulation	`MOVX.A #0,dst`
	`MOVX #0,dst`
	`MOVX.B #0,dst`

Description	The destination operand is cleared.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	RAM address-word TONI is cleared.

```
CLRX.A   TONI   ; 0 -> TONI
```

4.6.3.9 CMPX

CMPX.A	Compare source address-word and destination address-word
CMPX.[W]	Compare source word and destination word
CMPX.B	Compare source byte and destination byte

Syntax	`CMPX.A src,dst`
	`CMPX src,dst` or `CMPX.W src,dst`
	`CMPX.B src,dst`

Operation	(.not. src) + 1 + dst or dst – src
Description	The source operand is subtracted from the destination operand by adding the 1s complement of the source + 1 to the destination. The result affects only the status bits. Both operands may be located in the full address space.
Status Bits	N: Set if result is negative (src > dst), reset if positive (src ≤ dst) Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst) C: Set if there is a carry from the MSB, reset otherwise V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Compare EDE with a 20-bit constant 18000h. Jump to label TONI if EDE equals the constant.

```
CMPX.A   #018000h,EDE     ; Compare EDE with 18000h
JEQ      TONI             ; EDE contains 18000h
...                       ; Not equal
```

Example A table word pointed to by R5 (20-bit address) is compared with R7. Jump to label TONI if R7 contains a lower, signed, 16-bit number.

```
CMPX.W   @R5,R7           ; Compare two signed numbers
JL       TONI             ; R7 < @R5
...                       ; R7 >= @R5
```

Example A table byte pointed to by R5 (20-bit address) is compared to the input in I/O Port1. Jump to label TONI if the values are equal. The next table byte is addressed.

```
CMPX.B   @R5+,&P1IN       ; Compare P1 bits with table. R5 + 1
JEQ      TONI             ; Equal contents
...                       ; Not equal
```

Note: Use CMPA for the following two cases for better density and execution.

```
CMPA     Rsrc,Rdst
CMPA     #imm20,Rdst
```

4.6.3.10 DADCX

*** DADCX.A**	Add carry decimally to destination address-word
*** DADCX.[W]**	Add carry decimally to destination word
*** DADCX.B**	Add carry decimally to destination byte
Syntax	`DADCX.A dst`
	`DADCX dst` or `DADCX.W dst`
	`DADCX.B dst`
Operation	dst + C → dst (decimally)
Emulation	`DADDX.A #0,dst`
	`DADDX #0,dst`
	`DADDX.B #0,dst`
Description	The carry bit (C) is added decimally to the destination.
Status Bits	N: Set if MSB of result is 1 (address-word > 79999h, word > 7999h, byte > 79h), reset if MSB is 0
	Z: Set if result is zero, reset otherwise
	C: Set if the BCD result is too large (address-word > 99999h, word > 9999h, byte > 99h), reset otherwise
	V: Undefined
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 40-bit counter, pointed to by R12 and R13, is incremented decimally.

```
DADDX.A   #1,0(R12)     ; Increment lower 20 bits
DADCX.A   0(R13)        ; Add carry to upper 20 bits
```

4.6.3.11 DADDX

DADDX.A	Add source address-word and carry decimally to destination address-word
DADDX.[W]	Add source word and carry decimally to destination word
DADDX.B	Add source byte and carry decimally to destination byte

Syntax	`DADDX.A src,dst`
	`DADDX src,dst` or `DADDX.W src,dst`
	`DADDX.B src,dst`

Operation	src + dst + C → dst (decimally)
Description	The source operand and the destination operand are treated as two (.B), four (.W), or five (.A) binary coded decimals (BCD) with positive signs. The source operand and the carry bit C are added decimally to the destination operand. The source operand is not affected. The previous contents of the destination are lost. The result is not defined for non-BCD numbers. Both operands may be located in the full address space.
Status Bits	N: Set if MSB of result is 1 (address-word > 79999h, word > 7999h, byte > 79h), reset if MSB is 0. Z: Set if result is zero, reset otherwise C: Set if the BCD result is too large (address-word > 99999h, word > 9999h, byte > 99h), reset otherwise V: Undefined
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Decimal 10 is added to the 20-bit BCD counter DECCNTR located in two words.

```
DADDX.A   #10h,&DECCNTR     ; Add 10 to 20-bit BCD counter
```

Example The eight-digit BCD number contained in 20-bit addresses BCD and BCD+2 is added decimally to an eight-digit BCD number contained in R4 and R5 (BCD+2 and R5 contain the MSDs).

```
CLRC                        ; Clear carry
DADDX.W   BCD,R4            ; Add LSDs
DADDX.W   BCD+2,R5          ; Add MSDs with carry
JC        OVERFLOW          ; Result >99999999: go to error routine
...                         ;    Result ok
```

Example The two-digit BCD number contained in 20-bit address BCD is added decimally to a two-digit BCD number contained in R4.

```
CLRC                        ; Clear carry
DADDX.B   BCD,R4            ; Add BCD to R4 decimally.
                            ; R4: 000ddh
```

4.6.3.12 DECX

*** DECX.A**	Decrement destination address-word
*** DECX.[W]**	Decrement destination word
*** DECX.B**	Decrement destination byte
Syntax	`DECX.A dst`
	`DECX dst` or `DECX.W dst`
	`DECX.B dst`
Operation	dst – 1 → dst
Emulation	`SUBX.A #1,dst`
	`SUBX #1,dst`
	`SUBX.B #1,dst`
Description	The destination operand is decremented by one. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive
	Z: Set if dst contained 1, reset otherwise
	C: Reset if dst contained 0, set otherwise
	V: Set if an arithmetic overflow occurs, otherwise reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	RAM address-word TONI is decremented by one.

```
DECX.A   TONI     ; Decrement TONI
```

4.6.3.13 DECDX

*** DECDX.A**	Double-decrement destination address-word
*** DECDX.[W]**	Double-decrement destination word
*** DECDX.B**	Double-decrement destination byte

Syntax	`DECDX.A dst`
	`DECDX dst` or `DECDX.W dst`
	`DECDX.B dst`

Operation	dst – 2 → dst

Emulation	`SUBX.A #2,dst`
	`SUBX #2,dst`
	`SUBX.B #2,dst`

Description	The destination operand is decremented by two. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive Z: Set if dst contained 2, reset otherwise C: Reset if dst contained 0 or 1, set otherwise V: Set if an arithmetic overflow occurs, otherwise reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	RAM address-word TONI is decremented by two.

```
DECDX.A    TONI      ; Decrement TONI
```

4.6.3.14 INCX

*** INCX.A**	Increment destination address-word
*** INCX.[W]**	Increment destination word
*** INCX.B**	Increment destination byte

Syntax	`INCX.A dst`
	`INCX dst` or `INCX.W dst`
	`INCX.B dst`

Operation	dst + 1 → dst

Emulation	`ADDX.A #1,dst`
	`ADDX #1,dst`
	`ADDX.B #1,dst`

Description	The destination operand is incremented by one. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive Z: Set if dst contained 0FFFFFh, reset otherwise Set if dst contained 0FFFFh, reset otherwise Set if dst contained 0FFh, reset otherwise C: Set if dst contained 0FFFFFh, reset otherwise Set if dst contained 0FFFFh, reset otherwise Set if dst contained 0FFh, reset otherwise V: Set if dst contained 07FFFh, reset otherwise Set if dst contained 07FFFh, reset otherwise Set if dst contained 07Fh, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	RAM address-wordTONI is incremented by one.

```
INCX.A   TONI     ; Increment TONI (20-bits)
```

4.6.3.15 INCDX

*** INCDX.A**	Double-increment destination address-word
*** INCDX.[W]**	Double-increment destination word
*** INCDX.B**	Double-increment destination byte

Syntax	`INCDX.A dst`
	`INCDX dst` or `INCDX.W dst`
	`INCDX.B dst`

Operation	dst + 2 → dst

Emulation	`ADDX.A #2,dst`
	`ADDX #2,dst`
	`ADDX.B #2,dst`

Description	The destination operand is incremented by two. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive Z: Set if dst contained 0FFFFEh, reset otherwise Set if dst contained 0FFFEh, reset otherwise Set if dst contained 0FEh, reset otherwise C: Set if dst contained 0FFFFEh or 0FFFFFh, reset otherwise Set if dst contained 0FFFEh or 0FFFFh, reset otherwise Set if dst contained 0FEh or 0FFh, reset otherwise V: Set if dst contained 07FFFEh or 07FFFFh, reset otherwise Set if dst contained 07FFEh or 07FFFh, reset otherwise Set if dst contained 07Eh or 07Fh, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	RAM byte LEO is incremented by two; PC points to upper memory.

```
INCDX.B   LEO      ; Increment LEO by two
```

4.6.3.16 INVX

*** INVX.A** Invert destination

*** INVX.[W]** Invert destination

*** INVX.B** Invert destination

Syntax `INVX.A dst`

`INVX dst` or `INVX.W dst`

`INVX.B dst`

Operation .NOT.dst → dst

Emulation `XORX.A #0FFFFFh,dst`

`XORX #0FFFFh,dst`

`XORX.B #0FFh,dst`

Description The destination operand is inverted. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive

Z: Set if dst contained 0FFFFFh, reset otherwise
Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise

C: Set if result is not zero, reset otherwise (= .NOT. Zero)

V: Set if initial destination operand was negative, otherwise reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example 20-bit content of R5 is negated (2s complement).

```
INVX.A   R5       ; Invert R5
INCX.A   R5       ; R5 is now negated
```

Example Content of memory byte LEO is negated. PC is pointing to upper memory.

```
INVX.B   LEO      ; Invert LEO
INCX.B   LEO      ; MEM(LEO) is negated
```

4.6.3.17 MOVX

MOVX.A	Move source address-word to destination address-word
MOVX.[W]	Move source word to destination word
MOVX.B	Move source byte to destination byte

Syntax	`MOVX.A src,dst`
	`MOVX src,dst` or `MOVX.W src,dst`
	`MOVX.B src,dst`

Operation	src → dst
Description	The source operand is copied to the destination. The source operand is not affected. Both operands may be located in the full address space.
Status Bits	N: Not affected Z: Not affected C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Move a 20-bit constant 18000h to absolute address-word EDE

```
MOVX.A   #018000h,&EDE         ; Move 18000h to EDE
```

Example The contents of table EDE (word data, 20-bit addresses) are copied to table TOM. The length of the table is 030h words.

```
      MOVA     #EDE,R10                ; Prepare pointer (20-bit address)
Loop  MOVX.W   @R10+,TOM-EDE-2(R10)    ; R10 points to both tables.
                                       ; R10+2
      CMPA     #EDE+60h,R10            ; End of table reached?
      JLO      Loop                    ; Not yet
      ...                              ; Copy completed
```

Example The contents of table EDE (byte data, 20-bit addresses) are copied to table TOM. The length of the table is 020h bytes.

```
      MOVA     #EDE,R10                ; Prepare pointer (20-bit)
      MOV      #20h,R9                 ; Prepare counter
Loop  MOVX.W   @R10+,TOM-EDE-2(R10)    ; R10 points to both tables.
                                       ; R10+1
      DEC      R9                      ; Decrement counter
      JNZ      Loop                    ; Not yet done
      ...                              ; Copy completed
```

Ten of the 28 possible addressing combinations of the MOVX.A instruction can use the MOVA instruction. This saves two bytes and code cycles. Examples for the addressing combinations are:

```
MOVX.A   Rsrc,Rdst           MOVA   Rsrc,Rdst          ; Reg/Reg
MOVX.A   #imm20,Rdst         MOVA   #imm20,Rdst        ; Immediate/Reg
MOVX.A   &abs20,Rdst         MOVA   &abs20,Rdst        ; Absolute/Reg
MOVX.A   @Rsrc,Rdst          MOVA   @Rsrc,Rdst         ; Indirect/Reg
MOVX.A   @Rsrc+,Rdst         MOVA   @Rsrc+,Rdst        ; Indirect,Auto/Reg
MOVX.A   Rsrc,&abs20         MOVA   Rsrc,&abs20        ; Reg/Absolute
```

The next four replacements are possible only if 16-bit indexes are sufficient for the addressing:

```
MOVX.A   z20(Rsrc),Rdst      MOVA   z16(Rsrc),Rdst    ; Indexed/Reg
MOVX.A   Rsrc,z20(Rdst)      MOVA   Rsrc,z16(Rdst)    ; Reg/Indexed
MOVX.A   symb20,Rdst         MOVA   symb16,Rdst       ; Symbolic/Reg
MOVX.A   Rsrc,symb20         MOVA   Rsrc,symb16       ; Reg/Symbolic
```

4.6.3.18 POPM

POPM.A	Restore n CPU registers (20-bit data) from the stack	
POPM.[W]	Restore n CPU registers (16-bit data) from the stack	
Syntax	`POPM.A #n,Rdst`	1 ≤ n ≤ 16
	`POPM.W #n,Rdst` or `POPM #n,Rdst`	1 ≤ n ≤ 16

Operation POPM.A: Restore the register values from stack to the specified CPU registers. The SP is incremented by four for each register restored from stack. The 20-bit values from stack (two words per register) are restored to the registers.

POPM.W: Restore the 16-bit register values from stack to the specified CPU registers. The SP is incremented by two for each register restored from stack. The 16-bit values from stack (one word per register) are restored to the CPU registers.

Note : This instruction does not use the extension word.

Description POPM.A: The CPU registers pushed on the stack are moved to the extended CPU registers, starting with the CPU register (Rdst – n + 1). The SP is incremented by (n × 4) after the operation.

POPM.W: The 16-bit registers pushed on the stack are moved back to the CPU registers, starting with CPU register (Rdst – n + 1). The SP is incremented by (n × 2) after the instruction. The MSBs (Rdst.19:16) of the restored CPU registers are cleared.

Status Bits Status bits are not affected, except SR is included in the operation.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Restore the 20-bit registers R9, R10, R11, R12, R13 from the stack

```
POPM.A   #5,R13     ; Restore R9, R10, R11, R12, R13
```

Example Restore the 16-bit registers R9, R10, R11, R12, R13 from the stack.

```
POPM.W   #5,R13     ; Restore R9, R10, R11, R12, R13
```

4.6.3.19 PUSHM

PUSHM.A	Save n CPU registers (20-bit data) on the stack	
PUSHM.[W]	Save n CPU registers (16-bit words) on the stack	
Syntax	`PUSHM.A #n,Rdst`	1 ≤ n ≤ 16
	`PUSHM.W #n,Rdst` or `PUSHM #n,Rdst`	1 ≤ n ≤ 16

Operation PUSHM.A: Save the 20-bit CPU register values on the stack. The SP is decremented by four for each register stored on the stack. The MSBs are stored first (higher address).

PUSHM.W: Save the 16-bit CPU register values on the stack. The SP is decremented by two for each register stored on the stack.

Description PUSHM.A: The n CPU registers, starting with Rdst backwards, are stored on the stack. The SP is decremented by (n × 4) after the operation. The data (Rn.19:0) of the pushed CPU registers is not affected.

PUSHM.W: The n registers, starting with Rdst backwards, are stored on the stack. The SP is decremented by (n × 2) after the operation. The data (Rn.19:0) of the pushed CPU registers is not affected.

Note : This instruction does not use the extension word.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example Save the five 20-bit registers R9, R10, R11, R12, R13 on the stack

```
PUSHM.A   #5,R13     ; Save R13, R12, R11, R10, R9
```

Example Save the five 16-bit registers R9, R10, R11, R12, R13 on the stack

```
PUSHM.W   #5,R13     ; Save R13, R12, R11, R10, R9
```

4.6.3.20 POPX

*** POPX.A**	Restore single address-word from the stack
*** POPX.[W]**	Restore single word from the stack
*** POPX.B**	Restore single byte from the stack

Syntax	`POPX.A dst`
	 `POPX dst` or `POPX.W dst`
	 `POPX.B dst`

Operation	Restore the 8-, 16-, 20-bit value from the stack to the destination. 20-bit addresses are possible. The SP is incremented by two (byte and word operands) and by four (address-word operand).
Emulation	`MOVX(.B,.A) @SP+,dst`
Description	The item on TOS is written to the destination operand. Register mode, Indexed mode, Symbolic mode, and Absolute mode are possible. The SP is incremented by two or four. Note: the SP is incremented by two also for byte operations.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Write the 16-bit value on TOS to the 20-bit address &EDE

```
POPX.W   &EDE     ; Write word to address EDE
```

Example Write the 20-bit value on TOS to R9

```
POPX.A   R9     ; Write address-word to R9
```

4.6.3.21 PUSHX

PUSHX.A	Save single address-word to the stack
PUSHX.[W]	Save single word to the stack
PUSHX.B	Save single byte to the stack
Syntax	`PUSHX.A src`
	`PUSHX src` or `PUSHX.W src`
	`PUSHX.B src`
Operation	Save the 8-, 16-, 20-bit value of the source operand on the TOS. 20-bit addresses are possible. The SP is decremented by two (byte and word operands) or by four (address-word operand) before the write operation.
Description	The SP is decremented by two (byte and word operands) or by four (address-word operand). Then the source operand is written to the TOS. All seven addressing modes are possible for the source operand.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Save the byte at the 20-bit address &EDE on the stack

```
PUSHX.B   &EDE     ; Save byte at address EDE
```

Example Save the 20-bit value in R9 on the stack.

```
PUSHX.A   R9       ; Save address-word in R9
```

4.6.3.22 RLAM

RLAM.A	Rotate left arithmetically the 20-bit CPU register content	
RLAM.[W]	Rotate left arithmetically the 16-bit CPU register content	
Syntax	`RLAM.A #n,Rdst`	1 ≤ n ≤ 4
	`RLAM.W #n,Rdst` or `RLAM #n,Rdst`	1 ≤ n ≤ 4

Operation C ← MSB ← MSB-1 .... LSB+1 ← LSB ← 0

Description The destination operand is shifted arithmetically left one, two, three, or four positions as shown in Figure 4-44. RLAM works as a multiplication (signed and unsigned) with 2, 4, 8, or 16. The word instruction RLAM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits
- N: Set if result is negative
  - .A: Rdst.19 = 1, reset if Rdst.19 = 0
  - .W: Rdst.15 = 1, reset if Rdst.15 = 0
- Z: Set if result is zero, reset otherwise
- C: Loaded from the MSB (n = 1), MSB-1 (n = 2), MSB-2 (n = 3), MSB-3 (n = 4)
- V: Undefined

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit operand in R5 is shifted left by three positions. It operates equal to an arithmetic multiplication by 8.

```
RLAM.A   #3,R5      ; R5 = R5 x 8
```

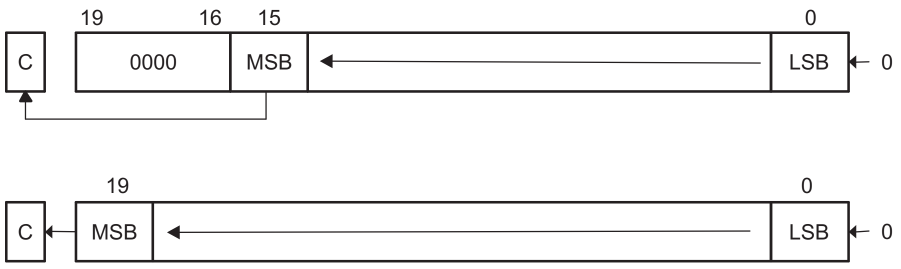

Figure 4-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A

4.6.3.23 RLAX

*** RLAX.A**	Rotate left arithmetically address-word
*** RLAX.[W]**	Rotate left arithmetically word
*** RLAX.B**	Rotate left arithmetically byte

Syntax	`RLAX.A dst`
	`RLAX dst` or `RLAX.W dst`
	`RLAX.B dst`

Operation	C ← MSB ← MSB-1 .... LSB+1 ← LSB ← 0

Emulation	`ADDX.A dst,dst`
	`ADDX dst,dst`
	`ADDX.B dst,dst`

Description	The destination operand is shifted left one position as shown in Figure 4-45. The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLAX instruction acts as a signed multiplication by 2.
Status Bits	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Loaded from the MSB V: Set if an arithmetic overflow occurs: the initial value is 040000h ≤ dst < 0C0000h; reset otherwise Set if an arithmetic overflow occurs: the initial value is 04000h ≤ dst < 0C000h; reset otherwise Set if an arithmetic overflow occurs: the initial value is 040h ≤ dst < 0C0h; reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 20-bit value in R7 is multiplied by 2

```
RLAX.A   R7     ; Shift left R7 (20-bit)
```

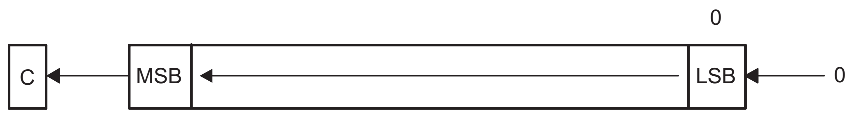

Figure 4-45. Destination Operand-Arithmetic Shift Left

4.6.3.24 RLCX

*** RLCX.A**	Rotate left through carry address-word
*** RLCX.[W]**	Rotate left through carry word
*** RLCX.B**	Rotate left through carry byte
Syntax	`RLCX.A dst`
	`RLCX dst` or `RLCX.W dst`
	`RLCX.B dst`
Operation	C ← MSB ← MSB-1 .... LSB+1 ← LSB ← C
Emulation	`ADDCX.A dst,dst`
	`ADDCX dst,dst`
	`ADDCX.B dst,dst`
Description	The destination operand is shifted left one position as shown in Figure 4-46. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).
Status Bits	N: Set if result is negative, reset if positive
	Z: Set if result is zero, reset otherwise
	C: Loaded from the MSB
	V: Set if an arithmetic overflow occurs: the initial value is 040000h ≤ dst < 0C0000h; reset otherwise
	Set if an arithmetic overflow occurs: the initial value is 04000h ≤ dst < 0C000h; reset otherwise
	Set if an arithmetic overflow occurs: the initial value is 040h ≤ dst < 0C0h; reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 20-bit value in R5 is shifted left one position.

```
RLCX.A   R5      ; (R5 x 2) + C -> R5
```

Example The RAM byte LEO is shifted left one position. PC is pointing to upper memory.

```
RLCX.B   LEO     ; RAM(LEO) x 2 + C -> RAM(LEO)
```

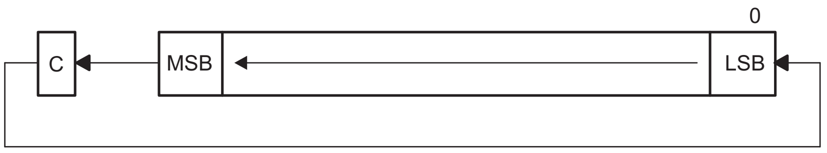

Figure 4-46. Destination Operand-Carry Left Shift

4.6.3.25 RRAM

RRAM.A	Rotate right arithmetically the 20-bit CPU register content	
RRAM.[W]	Rotate right arithmetically the 16-bit CPU register content	
Syntax	`RRAM.A #n,Rdst`	1 ≤ n ≤ 4
	`RRAM.W #n,Rdst` or `RRAM #n,Rdst`	1 ≤ n ≤ 4

Operation MSB → MSB → MSB–1 ... LSB+1 → LSB → C

Description The destination operand is shifted right arithmetically by one, two, three, or four bit positions as shown in Figure 4-47. The MSB retains its value (sign). RRAM operates equal to a signed division by 2, 4, 8, or 16. The MSB is retained and shifted into MSB-1. The LSB+1 is shifted into the LSB, and the LSB is shifted into the carry bit C. The word instruction RRAM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits
N: Set if result is negative
.A: Rdst.19 = 1, reset if Rdst.19 = 0
.W: Rdst.15 = 1, reset if Rdst.15 = 0
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB (n = 1), LSB+1 (n = 2), LSB+2 (n = 3), or LSB+3 (n = 4)
V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The signed 20-bit number in R5 is shifted arithmetically right two positions.

```
RRAM.A   #2,R5          ; R5/4 -> R5
```

Example The signed 20-bit value in R15 is multiplied by 0.75. (0.5 + 0.25) × R15.

```
PUSHM.A   #1,R15        ; Save extended R15 on stack
RRAM.A    #1,R15        ; R15 y 0.5 -> R15
ADDX.A    @SP+,R15      ; R15 y 0.5 + R15 = 1.5 y R15 -> R15
RRAM.A    #1,R15        ; (1.5 y R15) y 0.5 = 0.75 y R15 -> R15
```

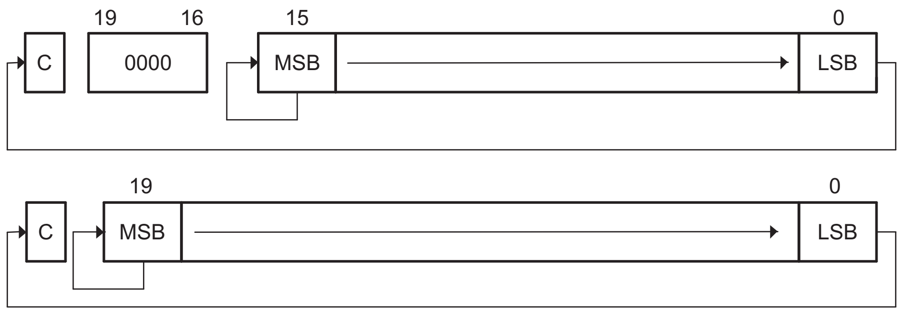

Figure 4-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A

4.6.3.26 RRAX

RRAX.A	Rotate right arithmetically the 20-bit operand
RRAX.[W]	Rotate right arithmetically the 16-bit operand
RRAX.B	Rotate right arithmetically the 8-bit operand

Syntax

```
RRAX.A Rdst
RRAX.W Rdst
RRAX Rdst
RRAX.B Rdst

RRAX.A dst
RRAX dst or                     RRAX.W dst
RRAX.B dst
```

Operation MSB → MSB → MSB–1 ... LSB+1 → LSB → C

Description Register mode for the destination: the destination operand is shifted right by one bit position as shown in Figure 4-48. The MSB retains its value (sign). The word instruction RRAX.W clears the bits Rdst.19:16, the byte instruction RRAX.B clears the bits Rdst.19:8. The MSB retains its value (sign), the LSB is shifted into the carry bit. RRAX here operates equal to a signed division by 2.

All other modes for the destination: the destination operand is shifted right arithmetically by one bit position as shown in Figure 4-49. The MSB retains its value (sign), the LSB is shifted into the carry bit. RRAX here operates equal to a signed division by 2. All addressing modes, with the exception of the Immediate mode, are possible in the full memory.

Status Bits

N: Set if result is negative, reset if positive
.A: dst.19 = 1, reset if dst.19 = 0
.W: dst.15 = 1, reset if dst.15 = 0
.B: dst.7 = 1, reset if dst.7 = 0

Z: Set if result is zero, reset otherwise

C: Loaded from the LSB

V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The signed 20-bit number in R5 is shifted arithmetically right four positions.

```
RPT      #4
RRAX.A   R5         ; R5/16 -> R5
```

Example The signed 8-bit value in EDE is multiplied by 0.5.

```
RRAX.B    &EDE      ; EDE/2 -> EDE
```

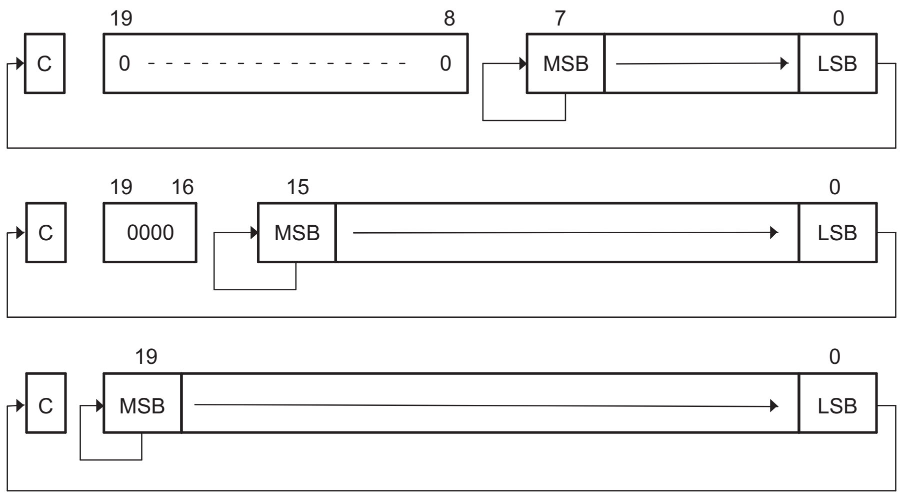

Figure 4-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode

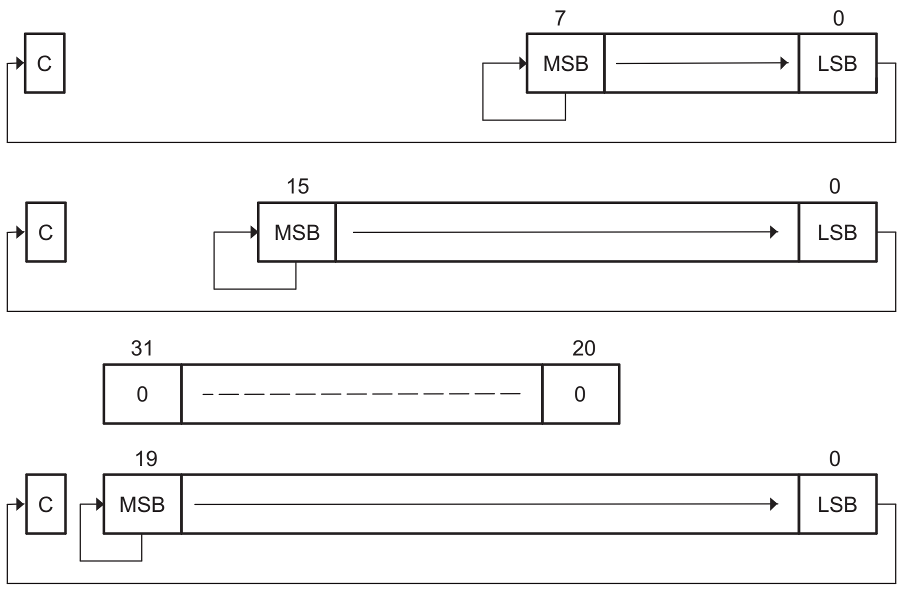

Figure 4-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode

4.6.3.27 RRCM

RRCM.A	Rotate right through carry the 20-bit CPU register content	
RRCM.[W]	Rotate right through carry the 16-bit CPU register content	
Syntax	`RRCM.A #n,Rdst`	1 ≤ n ≤ 4
	`RRCM.W #n,Rdst` or `RRCM #n,Rdst`	1 ≤ n ≤ 4

Operation C → MSB → MSB–1 ... LSB+1 → LSB → C

Description The destination operand is shifted right by one, two, three, or four bit positions as shown in Figure 4-50. The carry bit C is shifted into the MSB, the LSB is shifted into the carry bit. The word instruction RRCM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits

N:	Set if result is negative
	.A: Rdst.19 = 1, reset if Rdst.19 = 0
	.W: Rdst.15 = 1, reset if Rdst.15 = 0
Z:	Set if result is zero, reset otherwise
C:	Loaded from the LSB (n = 1), LSB+1 (n = 2), LSB+2 (n = 3), or LSB+3 (n = 4)
V:	Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The address-word in R5 is shifted right by three positions. The MSB–2 is loaded with 1.

```
SETC                    ; Prepare carry for MSB-2
RRCM.A    #3,R5         ; R5 = R5 » 3 + 20000h
```

Example The word in R6 is shifted right by two positions. The MSB is loaded with the LSB. The MSB–1 is loaded with the contents of the carry flag.

```
RRCM.W    #2,R6         ; R6 = R6 » 2. R6.19:16 = 0
```

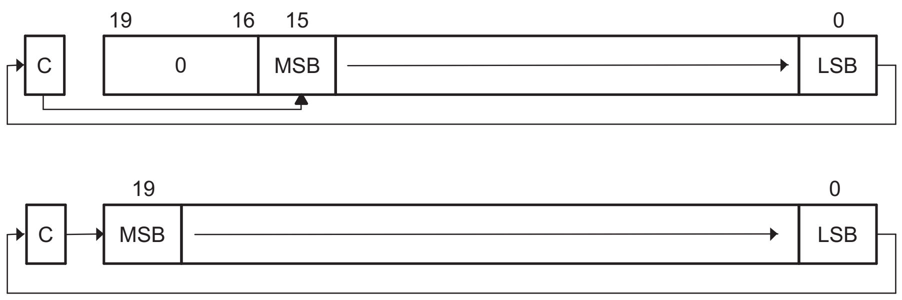

Figure 4-50. Rotate Right Through Carry RRCM[.W] and RRCM.A

4.6.3.28 RRCX

RRCX.A	Rotate right through carry the 20-bit operand
RRCX.[W]	Rotate right through carry the 16-bit operand
RRCX.B	Rotate right through carry the 8-bit operand

Syntax

```
RRCX.A Rdst
RRCX.W Rdst
RRCX Rdst
RRCX.B Rdst
RRCX.A dst
```

`RRCX dst` or `RRCX.W dst`

```
RRCX.B dst
```

Operation C → MSB → MSB–1 ... LSB+1 → LSB → C

Description Register mode for the destination: the destination operand is shifted right by one bit position as shown in Figure 4-51. The word instruction RRCX.W clears the bits Rdst.19:16, the byte instruction RRCX.B clears the bits Rdst.19:8. The carry bit C is shifted into the MSB, the LSB is shifted into the carry bit.

All other modes for the destination: the destination operand is shifted right by one bit position as shown in Figure 4-52. The carry bit C is shifted into the MSB, the LSB is shifted into the carry bit. All addressing modes, with the exception of the Immediate mode, are possible in the full memory.

Status Bits

N: Set if result is negative
.A: dst.19 = 1, reset if dst.19 = 0
.W: dst.15 = 1, reset if dst.15 = 0
.B: dst.7 = 1, reset if dst.7 = 0

Z: Set if result is zero, reset otherwise

C: Loaded from the LSB

V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit operand at address EDE is shifted right by one position. The MSB is loaded with 1.

```
SETC                ; Prepare carry for MSB
RRCX.A   EDE        ; EDE = EDE » 1 + 80000h
```

Example The word in R6 is shifted right by 12 positions.

```
RPT      #12
RRCX.W   R6       ; R6 = R6 » 12. R6.19:16 = 0
```

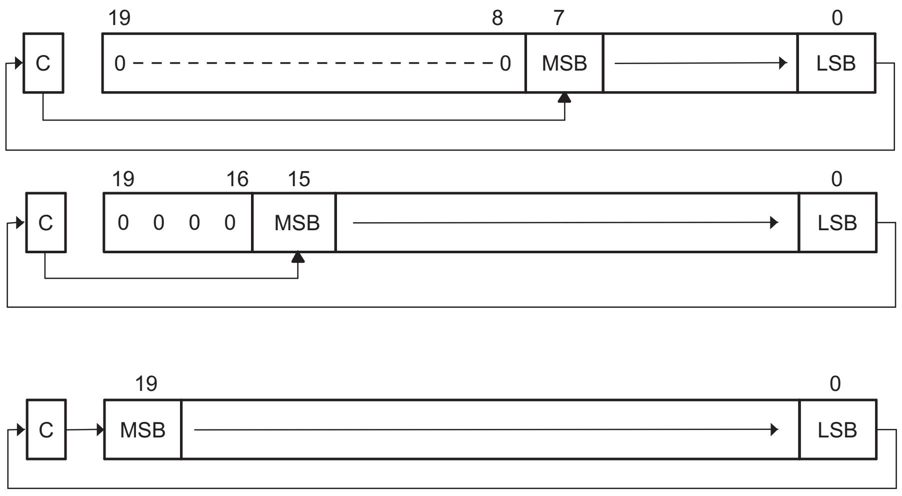

Figure 4-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode

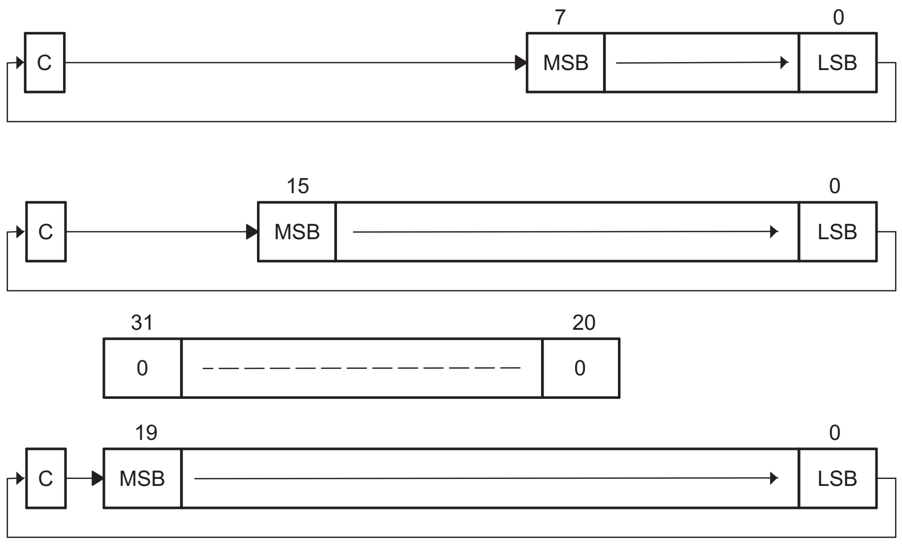

Figure 4-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode

4.6.3.29 RRUM

RRUM.A	Rotate right through carry the 20-bit CPU register content	
RRUM.[W]	Rotate right through carry the 16-bit CPU register content	
Syntax	`RRUM.A #n,Rdst`	1 ≤ n ≤ 4
	`RRUM.W #n,Rdst` or `RRUM #n,Rdst`	1 ≤ n ≤ 4

Operation 0 → MSB → MSB–1 ... LSB+1 → LSB → C

Description The destination operand is shifted right by one, two, three, or four bit positions as shown in Figure 4-53. Zero is shifted into the MSB, the LSB is shifted into the carry bit. RRUM works like an unsigned division by 2, 4, 8, or 16. The word instruction RRUM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits
- N: Set if result is negative
  - .A: Rdst.19 = 1, reset if Rdst.19 = 0
  - .W: Rdst.15 = 1, reset if Rdst.15 = 0
- Z: Set if result is zero, reset otherwise
- C: Loaded from the LSB (n = 1), LSB+1 (n = 2), LSB+2 (n = 3), or LSB+3 (n = 4)
- V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The unsigned address-word in R5 is divided by 16.

```
RRUM.A   #4,R5      ; R5 = R5 » 4. R5/16
```

Example The word in R6 is shifted right by one bit. The MSB R6.15 is loaded with 0.

```
RRUM.W   #1,R6      ; R6 = R6/2. R6.19:15 = 0
```

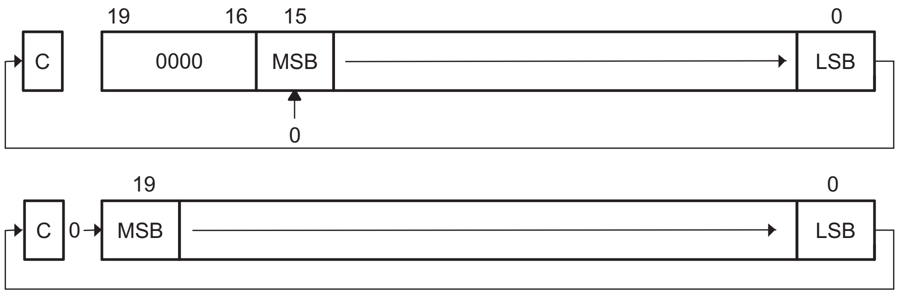

Figure 4-53. Rotate Right Unsigned RRUM[.W] and RRUM.A

4.6.3.30 RRUX

RRUX.A	Shift right unsigned the 20-bit CPU register content
RRUX.[W]	Shift right unsigned the 16-bit CPU register content
RRUX.B	Shift right unsigned the 8-bit CPU register content

Syntax

```
RRUX.A Rdst
RRUX.W Rdst
RRUX Rdst
RRUX.B Rdst
```

Operation C=0 → MSB → MSB–1 ... LSB+1 → LSB → C

Description RRUX is valid for register mode only: the destination operand is shifted right by one bit position as shown in Figure 4-54. The word instruction RRUX.W clears the bits Rdst.19:16. The byte instruction RRUX.B clears the bits Rdst.19:8. Zero is shifted into the MSB, the LSB is shifted into the carry bit.

Status Bits

N: Set if result is negative
.A: dst.19 = 1, reset if dst.19 = 0
.W: dst.15 = 1, reset if dst.15 = 0
.B: dst.7 = 1, reset if dst.7 = 0

Z: Set if result is zero, reset otherwise

C: Loaded from the LSB

V: Reset

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The word in R6 is shifted right by 12 positions.

```
RPT      #12
RRUX.W   R6       ; R6 = R6 » 12. R6.19:16 = 0
```

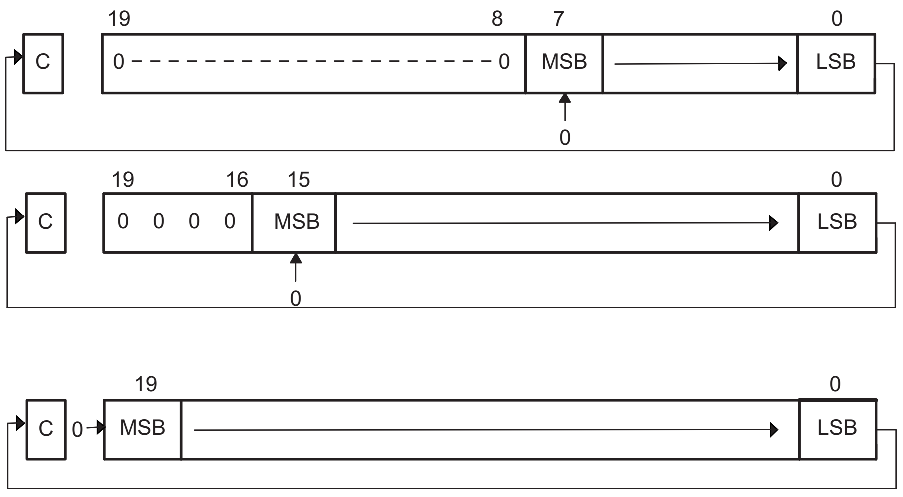

Figure 4-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode

4.6.3.31 SBCX

*** SBCX.A**	Subtract borrow (.NOT. carry) from destination address-word
*** SBCX.[W]**	Subtract borrow (.NOT. carry) from destination word
*** SBCX.B**	Subtract borrow (.NOT. carry) from destination byte
Syntax	`SBCX.A dst`
	`SBCX dst` or `SBCX.W dst`
	`SBCX.B dst`
Operation	dst + 0FFFFFh + C → dst
	dst + 0FFFFh + C → dst
	dst + 0FFh + C → dst
Emulation	`SBCX.A #0,dst`
	`SBCX #0,dst`
	`SBCX.B #0,dst`
Description	The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.
Status Bits	N: Set if result is negative, reset if positive
	Z: Set if result is zero, reset otherwise
	C: Set if there is a carry from the MSB of the result, reset otherwise Set to 1 if no borrow, reset if borrow
	V: Set if an arithmetic overflow occurs, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.

```
SUBX.B   @R13,0(R12)       ; Subtract LSDs
SBCX.B   1(R12)            ; Subtract carry from MSD
```

NOTE: Borrow implementation

The borrow is treated as a .NOT. carry:

Borrow	Carry Bit
Yes	0
No	1

4.6.3.32 SUBX

SUBX.A	Subtract source address-word from destination address-word
SUBX.[W]	Subtract source word from destination word
SUBX.B	Subtract source byte from destination byte

Syntax	`SUBX.A src,dst`
	`SUBX src,dst` or `SUBX.W src,dst`
	`SUBX.B src,dst`

Operation	(.not. src) + 1 + dst → dst or dst – src → dst
Description	The source operand is subtracted from the destination operand. This is done by adding the 1s complement of the source + 1 to the destination. The source operand is not affected. The result is written to the destination operand. Both operands may be located in the full address space.
Status Bits	N: Set if result is negative (src > dst), reset if positive (src ≤ dst) Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst) C: Set if there is a carry from the MSB, reset otherwise V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	A 20-bit constant 87654h is subtracted from EDE (LSBs) and EDE+2 (MSBs).

```
SUBX.A   #87654h,EDE     ; Subtract 87654h from EDE+2|EDE
```

Example A table word pointed to by R5 (20-bit address) is subtracted from R7. Jump to label TONI if R7 contains zero after the instruction. R5 is auto-incremented by two. R7.19:16 = 0.

```
SUBX.W   @R5+,R7         ; Subtract table number from R7. R5 + 2
JZ       TONI            ; R7 = @R5 (before subtraction)
...                      ; R7 <> @R5 (before subtraction)
```

Example Byte CNT is subtracted from the byte R12 points to in the full address space. Address of CNT is within PC ± 512 K.

```
SUBX.B   CNT,0(R12)      ; Subtract CNT from @R12
```

Note: Use SUBA for the following two cases for better density and execution.

```
SUBX.A   Rsrc,Rdst
SUBX.A   #imm20,Rdst
```

4.6.3.33 SUBCX

SUBCX.A	Subtract source address-word with carry from destination address-word
SUBCX.[W]	Subtract source word with carry from destination word
SUBCX.B	Subtract source byte with carry from destination byte

Syntax	`SUBCX.A src,dst`
	`SUBCX src,dst` or `SUBCX.W src,dst`
	`SUBCX.B src,dst`

Operation	(.not. src) + C + dst → dst or dst – (src – 1) + C → dst
Description	The source operand is subtracted from the destination operand. This is made by adding the 1s complement of the source + carry to the destination. The source operand is not affected, the result is written to the destination operand. Both operands may be located in the full address space.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB, reset otherwise V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow).
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	A 20-bit constant 87654h is subtracted from R5 with the carry from the previous instruction.

```
SUBCX.A   #87654h,R5      ; Subtract 87654h + C from R5
```

Example A 48-bit number (3 words) pointed to by R5 (20-bit address) is subtracted from a 48-bit counter in RAM, pointed to by R7. R5 auto-increments to point to the next 48-bit number.

```
SUBX.W    @R5+,0(R7)      ; Subtract LSBs. R5 + 2
SUBCX.W   @R5+,2(R7)      ; Subtract MIDs with C. R5 + 2
SUBCX.W   @R5+,4(R7)      ; Subtract MSBs with C. R5 + 2
```

Example Byte CNT is subtracted from the byte R12 points to. The carry of the previous instruction is used. 20-bit addresses.

```
SUBCX.B   &CNT,0(R12)     ; Subtract byte CNT from @R12
```

4.6.3.34 SWPBX

SWPBX.A	Swap bytes of lower word
SWPBX.[W]	Swap bytes of word
Syntax	`SWPBX.A dst`
	`SWPBX dst` or `SWPBX.W dst`
Operation	dst.15:8 ↔ dst.7:0
Description	Register mode: Rn.15:8 are swapped with Rn.7:0. When the .A extension is used, Rn.19:16 are unchanged. When the .W extension is used, Rn.19:16 are cleared. Other modes: When the .A extension is used, bits 31:20 of the destination address are cleared, bits 19:16 are left unchanged, and bits 15:8 are swapped with bits 7:0. When the .W extension is used, bits 15:8 are swapped with bits 7:0 of the addressed word.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Exchange the bytes of RAM address-word EDE

```
MOVX.A    #23456h,&EDE      ; 23456h -> EDE
SWPBX.A   EDE               ; 25634h -> EDE
```

Example Exchange the bytes of R5

```
MOVA      #23456h,R5        ; 23456h -> R5
SWPBX.W   R5                ; 05634h -> R5
```

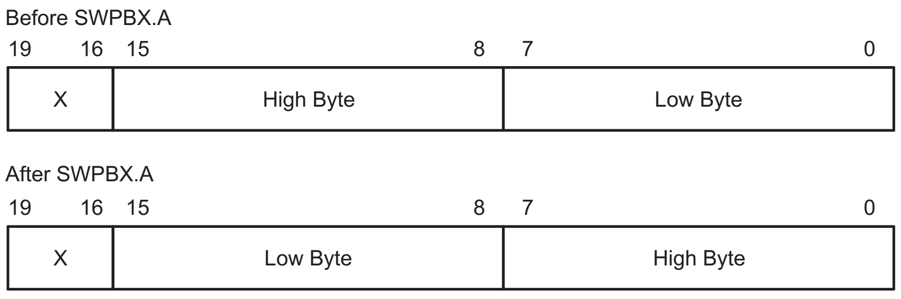

Figure 4-55. Swap Bytes SWPBX.A Register Mode

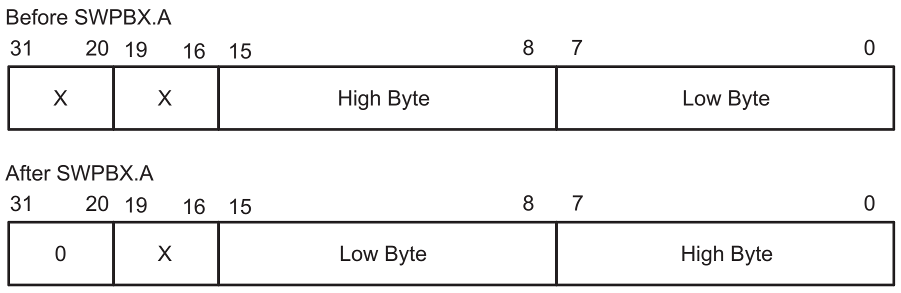

Figure 4-56. Swap Bytes SWPBX.A In Memory

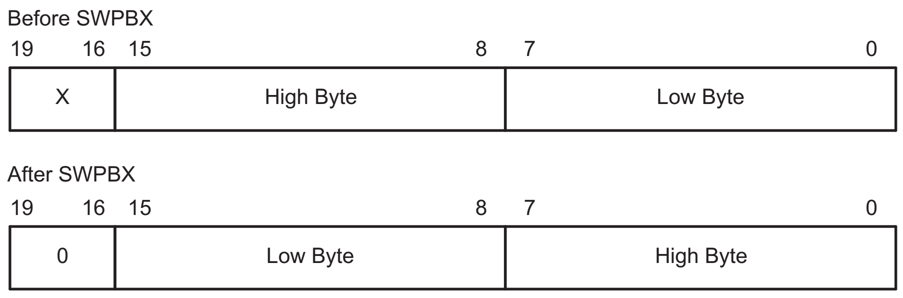

Figure 4-57. Swap Bytes SWPBX[.W] Register Mode

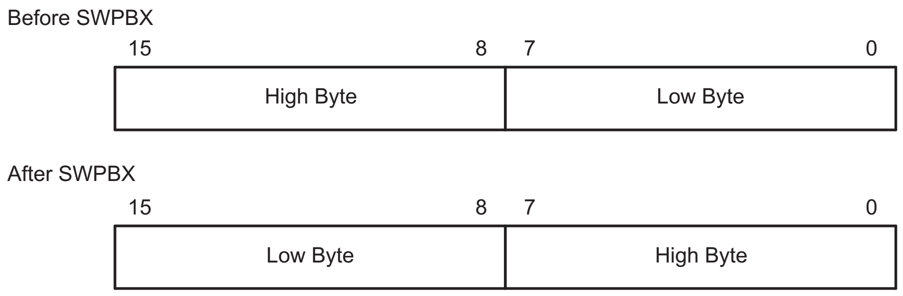

Figure 4-58. Swap Bytes SWPBX[.W] In Memory

4.6.3.35 SXTX

SXTX.A	Extend sign of lower byte to address-word
SXTX.[W]	Extend sign of lower byte to word
Syntax	`SXTX.A dst`
	`SXTX dst` or `SXTX.W dst`
Operation	dst.7 → dst.15:8, Rdst.7 → Rdst.19:8 (Register mode)
Description	Register mode: The sign of the low byte of the operand (Rdst.7) is extended into the bits Rdst.19:8.
	Other modes: SXTX.A: the sign of the low byte of the operand (dst.7) is extended into dst.19:8. The bits dst.31:20 are cleared.
	SXTX[.W]: the sign of the low byte of the operand (dst.7) is extended into dst.15:8.
Status Bits	N: Set if result is negative, reset otherwise
	Z: Set if result is zero, reset otherwise
	C: Set if result is not zero, reset otherwise (C = .not.Z)
	V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The signed 8-bit data in EDE.7:0 is sign extended to 20 bits: EDE.19:8. Bits 31:20 located in EDE+2 are cleared.

```
SXTX.A      &EDE            ; Sign extended EDE -> EDE+2/EDE
```

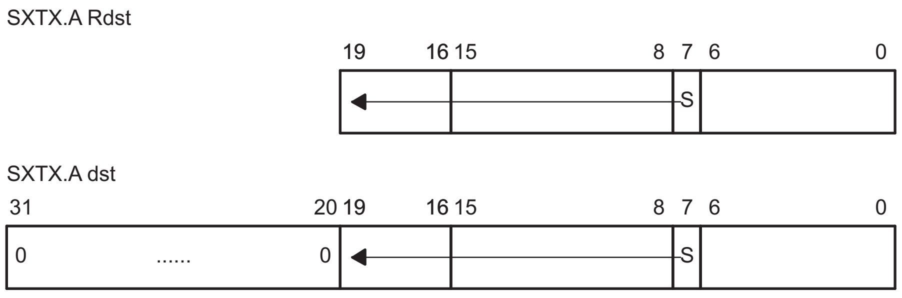

Figure 4-59. Sign Extend SXTX.A

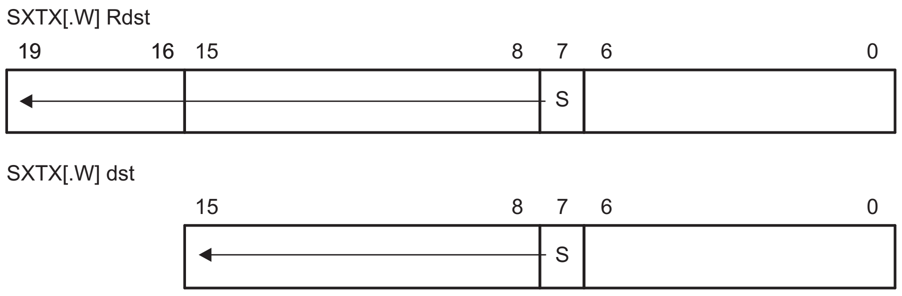

Figure 4-60. Sign Extend SXTX[.W]

4.6.3.36 TSTX

*** TSTX.A**	Test destination address-word
*** TSTX.[W]**	Test destination word
*** TSTX.B**	Test destination byte
Syntax	`TSTX.A dst`
	`TSTX dst` or `TSTX.W dst`
	`TSTX.B dst`
Operation	dst + 0FFFFFh + 1
	dst + 0FFFFh + 1
	dst + 0FFh + 1
Emulation	`CMPX.A #0,dst`
	`CMPX #0,dst`
	`CMPX.B #0,dst`
Description	The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.
Status Bits	N: Set if destination is negative, reset if positive
	Z: Set if destination contains zero, reset otherwise
	C: Set
	V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	RAM byte LEO is tested; PC is pointing to upper memory. If it is negative, continue at LEONEG; if it is positive but not zero, continue at LEOPOS.

```
          TSTX.B   LEO           ; Test LEO
          JN       LEONEG        ; LEO is negative
          JZ       LEOZERO       ; LEO is zero
LEOPOS    ......                 ; LEO is positive but not zero
LEONEG    ......                 ; LEO is negative
LEOZERO   ......                 ; LEO is zero
```

4.6.3.37 XORX

XORX.A	Exclusive OR source address-word with destination address-word
XORX.[W]	Exclusive OR source word with destination word
XORX.B	Exclusive OR source byte with destination byte

Syntax	`XORX.A src,dst`
	`XORX src,dst` or `XORX.W src,dst`
	`XORX.B src,dst`

Operation	src .xor. dst → dst
Description	The source and destination operands are exclusively ORed. The result is placed into the destination. The source operand is not affected. The previous contents of the destination are lost. Both operands may be located in the full address space.
Status Bits	N: Set if result is negative (MSB = 1), reset if positive (MSB = 0) Z: Set if result is zero, reset otherwise C: Set if result is not zero, reset otherwise (carry = .not. Zero) V: Set if both operands are negative (before execution), reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Toggle bits in address-word CNTR (20-bit data) with information in address-word TONI (20-bit address)

```
XORX.A   TONI,&CNTR      ; Toggle bits in CNTR
```

Example A table word pointed to by R5 (20-bit address) is used to toggle bits in R6.

```
XORX.W   @R5,R6          ; Toggle bits in R6. R6.19:16 = 0
```

Example Reset to zero those bits in the low byte of R7 that are different from the bits in byte EDE (20-bit address)

```
XORX.B   EDE,R7          ; Set different bits to 1 in R7
INV.B    R7              ; Invert low byte of R7. R7.19:8 = 0.
```

4.6.4 Address Instructions

MSP430X address instructions are instructions that support 20-bit operands but have restricted addressing modes. The addressing modes are restricted to the Register mode and the Immediate mode, except for the MOVA instruction. Restricting the addressing modes removes the need for the additional extension-word op-code improving code density and execution time. The MSP430X address instructions are listed and described in the following pages.

4.6.4.1 ADDA

ADDA	Add 20-bit source to a 20-bit destination register

Syntax	`ADDA Rsrc,Rdst`
	`ADDA #imm20,Rdst`

Operation	src + Rdst → Rdst
Description	The 20-bit source operand is added to the 20-bit destination CPU register. The previous contents of the destination are lost. The source operand is not affected.
Status Bits	N: Set if result is negative (Rdst.19 = 1), reset if positive (Rdst.19 = 0) Z: Set if result is zero, reset otherwise C: Set if there is a carry from the 20-bit result, reset otherwise V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	R5 is increased by 0A4320h. The jump to TONI is performed if a carry occurs.

```
ADDA   #0A4320h,R5      ; Add A4320h to 20-bit R5
JC     TONI             ; Jump on carry
...                     ; No carry occurred
```

4.6.4.2 BRA

*** BRA**	Branch to destination
Syntax	`BRA dst`
Operation	dst → PC
Emulation	`MOVA dst,PC`
Description	An unconditional branch is taken to a 20-bit address anywhere in the full address space. All seven source addressing modes can be used. The branch instruction is an address-word instruction. If the destination address is contained in a memory location X, it is contained in two ascending words: X (LSBs) and (X + 2) (MSBs).
Status Bits	N: Not affected
	Z: Not affected
	C: Not affected
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Examples	Examples for all addressing modes are given.

Immediate mode: Branch to label EDE located anywhere in the 20-bit address space or branch directly to address.

```
BRA      #EDE         ; MOVA   #imm20,PC
BRA      #01AA04h
```

Symbolic mode: Branch to the 20-bit address contained in addresses EXEC (LSBs) and EXEC+2 (MSBs). EXEC is located at the address (PC + X) where X is within +32 K. Indirect addressing.

```
BRA      EXEC         ; MOVA   z16(PC),PC
```

Note: If the 16-bit index is not sufficient, a 20-bit index may be used with the following instruction.

```
MOVX.A   EXEC,PC      ; 1M byte range with 20-bit index
```

Absolute mode: Branch to the 20-bit address contained in absolute addresses EXEC (LSBs) and EXEC+2 (MSBs). Indirect addressing.

```
BRA      &EXEC        ; MOVA   &abs20,PC
```

Register mode: Branch to the 20-bit address contained in register R5. Indirect R5.

```
BRA      R5           ; MOVA   R5,PC
```

Indirect mode: Branch to the 20-bit address contained in the word pointed to by register R5 (LSBs). The MSBs have the address (R5 + 2). Indirect, indirect R5.

```
BRA      @R5          ; MOVA   @R5,PC
```

Indirect, Auto-Increment mode: Branch to the 20-bit address contained in the words pointed to by register R5 and increment the address in R5 afterwards by 4. The next time the software flow uses R5 as a pointer, it can alter the program execution due to access to the next address in the table pointed to by R5. Indirect, indirect R5.

```
BRA      @R5+          ; MOVA   @R5+,PC. R5 + 4
```

Indexed mode: Branch to the 20-bit address contained in the address pointed to by register (R5 + X) (for example, a table with addresses starting at X). (R5 + X) points to the LSBs, (R5 + X + 2) points to the MSBs of the address. X is within R5 + 32 K. Indirect, indirect (R5 + X).

```
BRA      X(R5)         ; MOVA   z16(R5),PC
```

Note: If the 16-bit index is not sufficient, a 20-bit index X may be used with the following instruction:

```
MOVX.A   X(R5),PC      ; 1M byte range with 20-bit index
```

4.6.4.3 CALLA

CALLA	Call a subroutine
Syntax	`CALLA dst`
Operation	dst → tmp 20-bit dst is evaluated and stored SP – 2 → SP PC.19:16 → @SP updated PC with return address to TOS (MSBs) SP – 2 → SP PC.15:0 → @SP updated PC to TOS (LSBs) tmp → PC saved 20-bit dst to PC
Description	A subroutine call is made to a 20-bit address anywhere in the full address space. All seven source addressing modes can be used. The call instruction is an address-word instruction. If the destination address is contained in a memory location X, it is contained in two ascending words, X (LSBs) and (X + 2) (MSBs). Two words on the stack are needed for the return address. The return is made with the instruction RETA.
Status Bits	N: Not affected Z: Not affected C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Examples	Examples for all addressing modes are given.

Immediate mode: Call a subroutine at label EXEC or call directly an address.

```
CALLA   #EXEC          ; Start address EXEC
CALLA   #01AA04h       ; Start address 01AA04h
```

Symbolic mode: Call a subroutine at the 20-bit address contained in addresses EXEC (LSBs) and EXEC+2 (MSBs). EXEC is located at the address (PC + X) where X is within +32 K. Indirect addressing.

```
CALLA   EXEC           ; Start address at @EXEC. z16(PC)
```

Absolute mode: Call a subroutine at the 20-bit address contained in absolute addresses EXEC (LSBs) and EXEC+2 (MSBs). Indirect addressing.

```
CALLA   &EXEC          ; Start address at @EXEC
```

Register mode: Call a subroutine at the 20-bit address contained in register R5. Indirect R5.

```
CALLA   R5             ; Start address at @R5
```

Indirect mode: Call a subroutine at the 20-bit address contained in the word pointed to by register R5 (LSBs). The MSBs have the address (R5 + 2). Indirect, indirect R5.

```
CALLA   @R5            ; Start address at @R5
```

Indirect, Auto-Increment mode: Call a subroutine at the 20-bit address contained in the words pointed to by register R5 and increment the 20-bit address in R5 afterwards by 4. The next time the software flow uses R5 as a pointer, it can alter the program execution due to access to the next word address in the table pointed to by R5. Indirect, indirect R5.

```
CALLA   @R5+         ; Start address at @R5. R5 + 4
```

Indexed mode: Call a subroutine at the 20-bit address contained in the address pointed to by register (R5 + X); for example, a table with addresses starting at X. (R5 + X) points to the LSBs, (R5 + X + 2) points to the MSBs of the word address. X is within R5 + 32 K. Indirect, indirect (R5 + X).

```
CALLA   X(R5)        ; Start address at @(R5+X). z16(R5)
```

4.6.4.4 CLRA

*** CLRA**	Clear 20-bit destination register
Syntax	`CLRA Rdst`
Operation	0 → Rdst
Emulation	`MOVA #0,Rdst`
Description	The destination register is cleared.
Status Bits	Status bits are not affected.
Example	The 20-bit value in R10 is cleared.

```
        CLRA    R10       ; 0 -> R10
```

4.6.4.5 CMPA

CMPA Compare the 20-bit source with a 20-bit destination register

Syntax `CMPA Rsrc,Rdst`

`CMPA #imm20,Rdst`

Operation (.not. src) + 1 + Rdst or Rdst – src

Description The 20-bit source operand is subtracted from the 20-bit destination CPU register. This is made by adding the 1s complement of the source + 1 to the destination register. The result affects only the status bits.

Status Bits

N:	Set if result is negative (src > dst), reset if positive (src ≤ dst)
Z:	Set if result is zero (src = dst), reset otherwise (src ≠ dst)
C:	Set if there is a carry from the MSB, reset otherwise
V:	Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example A 20-bit immediate operand and R6 are compared. If they are equal, the program continues at label EQUAL.

```
CMPA    #12345h,R6      ; Compare R6 with 12345h
JEQ     EQUAL           ; R6 = 12345h
...                     ; Not equal
```

Example The 20-bit values in R5 and R6 are compared. If R5 is greater than (signed) or equal to R6, the program continues at label GRE.

```
CMPA    R6,R5           ; Compare R6 with R5 (R5 - R6)
JGE     GRE             ; R5 >= R6
...                     ; R5 < R6
```

4.6.4.6 DECDA

*** DECDA**	Double-decrement 20-bit destination register
Syntax	`DECDA Rdst`
Operation	Rdst – 2 → Rdst
Emulation	`SUBA #2,Rdst`
Description	The destination register is decremented by two. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive
	Z: Set if Rdst contained 2, reset otherwise
	C: Reset if Rdst contained 0 or 1, set otherwise
	V: Set if an arithmetic overflow occurs, otherwise reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 20-bit value in R5 is decremented by 2.

```
DECDA   R5      ; Decrement R5 by two
```

4.6.4.7 INCDA

*** INCDA**	Double-increment 20-bit destination register
Syntax	`INCDA Rdst`
Operation	Rdst + 2 → Rdst
Emulation	`ADDA #2,Rdst`
Description	The destination register is incremented by two. The original contents are lost.
Status Bits	N: Set if result is negative, reset if positive
	Z: Set if Rdst contained 0FFFFEh, reset otherwise
	Set if Rdst contained 0FFFEh, reset otherwise
	Set if Rdst contained 0FEh, reset otherwise
	C: Set if Rdst contained 0FFFFEh or 0FFFFFh, reset otherwise
	Set if Rdst contained 0FFFEh or 0FFFFh, reset otherwise
	Set if Rdst contained 0FEh or 0FFh, reset otherwise
	V: Set if Rdst contained 07FFFEh or 07FFFFh, reset otherwise
	Set if Rdst contained 07FFEh or 07FFFh, reset otherwise
	Set if Rdst contained 07Eh or 07Fh, reset otherwise
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 20-bit value in R5 is incremented by two.

```
INCDA   R5      ; Increment R5 by two
```

4.6.4.8 MOVA

MOVA Move the 20-bit source to the 20-bit destination

Syntax
```
MOVA Rsrc,Rdst
MOVA #imm20,Rdst
MOVA z16(Rsrc),Rdst
MOVA EDE,Rdst
MOVA &abs20,Rdst
MOVA @Rsrc,Rdst
MOVA @Rsrc+,Rdst
MOVA Rsrc,z16(Rdst)
MOVA Rsrc,&abs20
```

Operation src → Rdst

Rsrc → dst

Description The 20-bit source operand is moved to the 20-bit destination. The source operand is not affected. The previous content of the destination is lost.

Status Bits

N:	Not affected
Z:	Not affected
C:	Not affected
V:	Not affected

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Examples Copy 20-bit value in R9 to R8

```
MOVA    R9,R8          ; R9 -> R8
```

Write 20-bit immediate value 12345h to R12

```
MOVA    #12345h,R12    ; 12345h -> R12
```

Copy 20-bit value addressed by (R9 + 100h) to R8. Source operand in addresses (R9 + 100h) LSBs and (R9 + 102h) MSBs.

```
MOVA    100h(R9),R8       ; Index: + 32 K. 2 words transferred
```

Move 20-bit value in 20-bit absolute addresses EDE (LSBs) and EDE+2 (MSBs) to R12

```
MOVA    &EDE,R12          ; &EDE -> R12. 2 words transferred
```

Move 20-bit value in 20-bit addresses EDE (LSBs) and EDE+2 (MSBs) to R12. PC index ± 32 K.

```
MOVA    EDE,R12           ; EDE -> R12. 2 words transferred
```

Copy 20-bit value R9 points to (20 bit address) to R8. Source operand in addresses @R9 LSBs and @(R9 + 2) MSBs.

```
MOVA    @R9,R8            ; @R9 -> R8. 2 words transferred
```

Copy 20-bit value R9 points to (20 bit address) to R8. R9 is incremented by four afterwards. Source operand in addresses @R9 LSBs and @(R9 + 2) MSBs.

```
MOVA   @R9+,R8          ; @R9 -> R8. R9 + 4. 2 words transferred.
```

Copy 20-bit value in R8 to destination addressed by (R9 + 100h). Destination operand in addresses @(R9 + 100h) LSBs and @(R9 + 102h) MSBs.

```
MOVA   R8,100h(R9)      ; Index: +- 32 K. 2 words transferred
```

Move 20-bit value in R13 to 20-bit absolute addresses EDE (LSBs) and EDE+2 (MSBs)

```
MOVA   R13,&EDE         ; R13 -> EDE. 2 words transferred
```

Move 20-bit value in R13 to 20-bit addresses EDE (LSBs) and EDE+2 (MSBs). PC index ± 32 K.

```
MOVA   R13,EDE          ; R13 -> EDE. 2 words transferred
```

4.6.4.9 RETA

*** RETA**	Return from subroutine
Syntax	`RETA`
Operation	@SP → PC.15:0 LSBs (15:0) of saved PC to PC.15:0
	SP + 2 → SP
	@SP → PC.19:16 MSBs (19:16) of saved PC to PC.19:16
	SP + 2 → SP
Emulation	`MOVA @SP+,PC`
Description	The 20-bit return address information, pushed onto the stack by a CALLA instruction, is restored to the PC. The program continues at the address following the subroutine call. The SR bits SR.11:0 are not affected. This allows the transfer of information with these bits.
Status Bits	N: Not affected
	Z: Not affected
	C: Not affected
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Call a subroutine SUBR from anywhere in the 20-bit address space and return to the address after the CALLA

```
        CALLA    #SUBR      ; Call subroutine starting at SUBR
        ...                 ; Return by RETA to here
SUBR    PUSHM.A  #2,R14     ; Save R14 and R13 (20 bit data)
        ...                 ; Subroutine code
        POPM.A   #2,R14     ; Restore R13 and R14 (20 bit data)
        RETA                ; Return (to full address space)
```

4.6.4.10 SUBA

SUBA Subtract 20-bit source from 20-bit destination register

Syntax `SUBA Rsrc,Rdst`

`SUBA #imm20,Rdst`

Operation (.not.src) + 1 + Rdst → Rdst   or   Rdst – src → Rdst

Description The 20-bit source operand is subtracted from the 20-bit destination register. This is made by adding the 1s complement of the source + 1 to the destination. The result is written to the destination register, the source is not affected.

Status Bits

N:	Set if result is negative (src > dst), reset if positive (src ≤ dst)
Z:	Set if result is zero (src = dst), reset otherwise (src ≠ dst)
C:	Set if there is a carry from the MSB (Rdst.19), reset otherwise
V:	Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Example The 20-bit value in R5 is subtracted from R6. If a carry occurs, the program continues at label TONI.

```
SUBA  R5,R6      ; R6 - R5 -> R6
JC    TONI       ; Carry occurred
...              ; No carry
```

4.6.4.11 TSTA

*** TSTA**	Test 20-bit destination register
Syntax	`TSTA Rdst`
Operation	dst + 0FFFFFh + 1 dst + 0FFFFh + 1 dst + 0FFh + 1
Emulation	`CMPA #0,Rdst`
Description	The destination register is compared with zero. The status bits are set according to the result. The destination register is not affected.
Status Bits	N: Set if destination register is negative, reset if positive Z: Set if destination register contains zero, reset otherwise C: Set V: Reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 20-bit value in R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
         TSTA   R7           ; Test R7
         JN     R7NEG        ; R7 is negative
         JZ     R7ZERO       ; R7 is zero
R7POS    ......              ; R7 is positive but not zero
R7NEG    ......              ; R7 is negative
R7ZERO   ......              ; R7 is zero
```