EEE 204
Assembly Language Programming on Embedded Processor

Asst. Prof. Dr Mahmut AYKAÇ

CHAPTER 4

Assembly Language Programming

Programs are written using a programming language with specific rules of syntax. These languages are found in three main levels:

a) Machine language,

b) Assembly language, and

c) Level language

The complete MSP430 instruction set consists of 27 core instructions and 24 emulated instructions. The core instructions are instructions that have unique op-codes decoded by the CPU. The emulated instructions are instructions that make code easier to write and read, but do not have op-codes themselves, instead they are replaced automatically by the assembler with an equivalent core instruction.

Opcode: The machine language code which defines the operation to be performed.

Assembly Language Programming

Mnemonic		Description		V	N	Z	C
ADC(.B) (1)	dst	Add C to destination	dst + C → dst	*	*	*	*
ADD(.B)	src,dst	Add source to destination	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	Add source and C to destination	src + dst + C → dst	*	*	*	*
AND(.B)	src,dst	AND source and destination	src .and. dst → dst	0	*	*	*
BIC(.B)	src,dst	Clear bits in destination	not.src .and. dst → dst	-	-	-	-
BIS(.B)	src,dst	Set bits in destination	src .or. dst → dst	-	-	-	-
BIT(.B)	src,dst	Test bits in destination	src .and. dst	0	*	*	*
BR (1)	dst	Branch to destination	dst → PC	-	-	-	-
CALL	dst	Call destination	PC+2 → stack, dst → PC	-	-	-	-
CLR(.B) (1)	dst	Clear destination	0 → dst	-	-	-	-
CLRC (1)		Clear C	0 → C	-	-	-	0
CLRN (1)		Clear N	0 → N	-	0	-	-
CLRZ (1)		Clear Z	0 → Z	-	-	0	-
CMP(.B)	src,dst	Compare source and destination	dst - src	*	*	*	*
DADC(.B) (1)	dst	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
DADD(.B)	src,dst	Add source and C decimally to dst	src + dst + C → dst (decimally)	*	*	*	*
DEC(.B) (1)	dst	Decrement destination	dst - 1 → dst	*	*	*	*

*	The status bit is affected
–	The status bit is not affected
0	The status bit is cleared
1	The status bit is set

(1) Emulated Instruction

Assembly Language Programming

Mnemonic		Description		V	N	Z	C
DECD(.B) [1]	dst	Double-decrement destination	dst - 2 → dst	*	*	*	*
DINT [1]		Disable interrupts	0 → GIE	-	-	-	-
EINT [1]		Enable interrupts	1 → GIE	-	-	-	-
INC(.B) [1]	dst	Increment destination	dst +1 → dst	*	*	*	*
INCD(.B) [1]	dst	Double-increment destination	dst+2 → dst	*	*	*	*
INV(.B) [1]	dst	Invert destination	.not.dst → dst	*	*	*	*
JC/JHS	label	Jump if C set/Jump if higher or same		-	-	-	-
JEQ/JZ	label	Jump if equal/Jump if Z set		-	-	-	-
JGE	label	Jump if greater or equal		-	-	-	-
JL	label	Jump if less		-	-	-	-
JMP	label	Jump	PC + 2 × offset → PC	-	-	-	-
JN	label	Jump if N set		-	-	-	-
JNC/JLO	label	Jump if C not set/Jump if lower		-	-	-	-
JNE/JNZ	label	Jump if not equal/Jump if Z not set		-	-	-	-
MOV(.B)	src,dst	Move source to destination	src → dst	-	-	-	-
NOP [2]		No operation		-	-	-	-
POP(.B) [2]	dst	Pop item from stack to destination	@SP → dst, SP+2 → SP	-	-	-	-
PUSH(.B)	src	Push source onto stack	SP - 2 → SP, src → @SP	-	-	-	-
RET [2]		Return from subroutine	@SP → PC, SP + 2 → SP	-	-	-	-
RETI		Return from interrupt		*	*	*	*
RLA(.B) [2]	dst	Rotate left arithmetically		*	*	*	*
RLC(.B) [2]	dst	Rotate left through C		*	*	*	*
RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
RRC(.B)	dst	Rotate right through C		*	*	*	*
SBC(.B) [2]	dst	Subtract not(C) from destination	dst + 0FFFFh + C → dst	*	*	*	*
SETC [2]		Set C	1 → C	-	-	-	1
SETN [2]		Set N	1 → N	-	1	-	-
SETZ [2]		Set Z	1 → Z	-	-	1	-
SUB(.B)	src,dst	Subtract source from destination	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	Subtract source and not(C) from dst	dst + .not.src + C → dst	*	*	*	*
SWPB	dst	Swap bytes		-	-	-	-
SXT	dst	Extend sign		0	*	*	*
TST(.B) [2]	dst	Test destination	dst + 0FFFFh + 1	0	*	*	1
XOR(.B)	src,dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

* The status bit is affected

– The status bit is not affected

0 The status bit is cleared

1 The status bit is set

(2) Emulated Instruction

Assembly Language Programming

There are three core-instruction formats:

- Dual-operand
- Single-operand
- Jump

All single-operand and dual-operand instructions can be byte or word instructions by using **`.B or .W`** extensions. Byte instructions are used to access byte data or byte peripherals. Word instructions are used to access word data or word peripherals. **If no extension is used, the instruction is a word instruction.**

MOV Instruction

MOV(.B or .W) src,dst; dst←src, move source to destination

The source is copied to the destination. The value in source is preserved. Type of source and destination may vary according to the addressing mode.

Examples:

```
MOV #00FAh, R10; load constant FAh into R10

MOV @R12, R4    ;move the content of memory address indexed by the content of R12 into R4
```

Addition

ADD(.B or .W) src,dst; dst←src + dst, add source to the destination

The source operand is added to the destination and the result is placed in the destination. The value in source is preserved.

Ex:

```
Add.b #3ah, r10; add 3AH to the content of R10 register
add @r4, r7; add the contents of the location pointed by R4 to R7
```

Note: In assembly language programming, it does not matter whether the letters are capital or lower case. That is why some letters are intentionally chosen to be lower case or capital. **It is not a case-sensitive programming language, not like C/C++.**

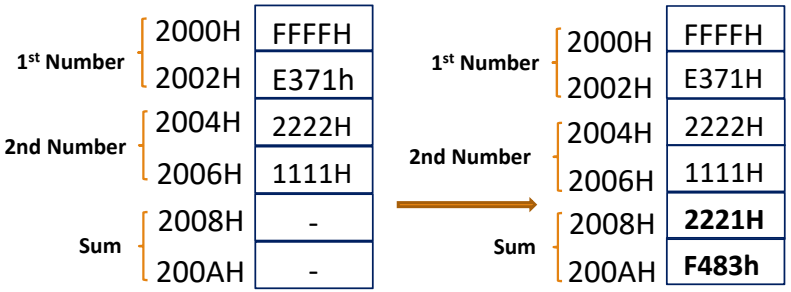
Addition

ADDC(.B or .W) src,dst; dst←src + dst+C, add source and carry to the destination.

The instruction is useful when trying to perform addition on numbers that are larger than 16-bits.

Ex: Add two 32- bit numbers E371FFFFh and 11112222h, whose addresses are 2000h and 2004h.

```
mov.w #2000h, R4 ;Loading the registers for the address of E371FFFFh
mov.w #2004h, R5 ;Loading the registers for the address of 11112222h
mov.w #2008h, R6;Loading the registers for the address of the sum
mov.w 0(R4), R7 ;Taking the lower parts
mov.w 0(R5), R8 ;Taking the lower parts
add.w R7, R8 ;Adding the lower parts without carry
mov.w R8,0(R6) ;Saving the lower part of the sum
mov.w 2(R4), R7 ;Taking the higher parts
mov.w 2(R5), R8 ;Taking the higher parts
addc.w R7, R8 ;Adding the higher part with carry
mov.w R8,2(R6) ;Saving the higher part of the sum
```



Example…

Write an assembly language program that adds 785H and 683H and save the result into the memory location addressed by 33FFh address.

```
mov.w #0x785,r4 ;r4=0785h
mov.w #0x683,r5 ;r5=0683h
add.w r4,r5 ;r5=r4+r5
mov.w #0x33FF,r6 ;r6=33FFh
mov.w r5,0(r6) ;r5→(33FFH)
```

Subtraction

SUB (.B or.W)src, dst; dst←dst-src, The source is subtracted from the destination and the result is saved in destination. The value in source is preserved

```
mov #0x1234, r4
sub #4, r4; subtract 4 from the content of R4 register
```

Subtraction

What if the result is negative?? Let's make it clear with the similar example

Ex. Write the following program and observe the output.

```
mov.w #0x2345,r4 ;r4=2345h
mov.w #0x5789,r5 ;r5=5789h
sub r5,r4 ;r4=r4-r5
```

N | 1 | Negative flag bit in status register becomes 1 since the result isnegative!

Subtraction

We can even make the byte operations even though R4 and R5 have words

Ex: Write the following program and observe the output.

```
mov.w #0x2345,r4 ;r4=2345h
mov.w #0x5789,r5 ;r5=5789h
sub.b r4,r5 ;r5=r5-r4
```

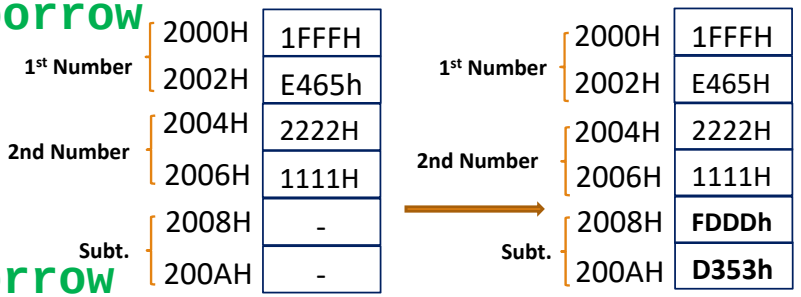
****If there is a byte operation (no matter what operation) on the registers, high byte of the source register becomes 0 because the related operation occurs on the low byte and the register is updated with the new content.**

Subtraction

SUBC (.B or.W)src, dst; It performs binary subtraction on the src and dst operands, but also subtracts not(C) from the status register that may have occurred from a prior subtraction dst←dst-src-not(C). This instruction is useful when trying to perform subtraction on numbers that are larger than 16-bits. **The Borrow is treated as a NOT carry.**

Ex: Subtract 11112222h from E4651FFFh, whose addresses are 2000h and 2004h.

```
mov.w #2000h, R4;Loading the register for the address of E4651FFFh
mov.w #2004h, R5;Loading the register for the address of 11112222h
mov.w #2008h, R6;Loading the register for the addresses of the subt.
mov.w 0(R4), R7 ;Taking the lower parts
mov.w 0(R5), R8 ;Taking the lower parts
sub.w R8, R7;Subtracting the lower part without borrow
mov.w R7,0(R6);Saving the lower part of the subt.
mov.w 2(R4), R7;Taking the higher parts
mov.w 2(R5), R8 ;Taking the higher parts
subc.w R8, R7;Subtracting the higher part with borrow
mov.w R7,2(R6);Saving the higher part of the subt.
```



Bitwise Logic

AND (.B or .W) src, dst ; The bits in the source and destination are ANDed and the result is saved in the destination. Source is not effected.

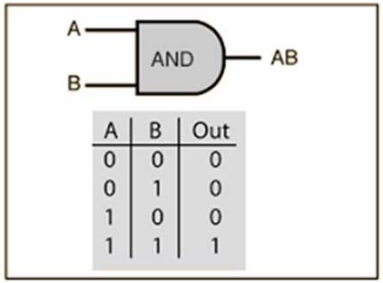
Ex: Assume contents of the registers and memory before any instruction as

R12 = 25A3h = 0010010110100011b, R15 = 8B94h = 1000101110010100b

```
mov.w #0x25A3, r12
mov.w #0x8b94, r15
and.w r15, r12
```

Operation:

0010 0101 1010 0011 (R12) AND
1000 1011 1001 0100 (R15) =
0000 0001 1000 0000



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

R12 | 25A3H | AND.W | R15 | 8B94 | → | R12 | 0180H

Bitwise Logic

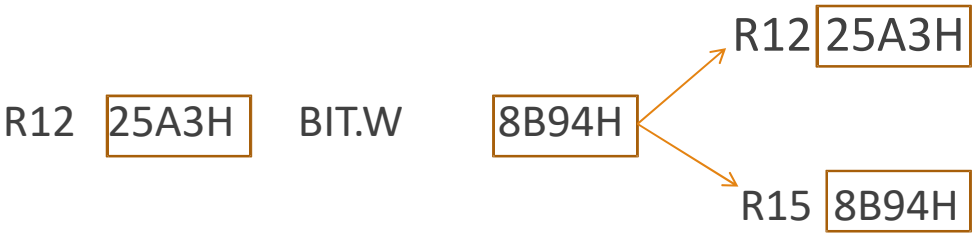
`BIT (.B or .W) src, dst` ; Identical to AND except the final value of destination. After BIT operation source and destination are preserved. **Only changing occurs in Status register bits such as N, C, V, and Z.**

Ex: Assume contents of the registers and memory before any instruction as

R12 = 25A3h = 0010010110100011, R15 = 8B94h = 1000101110010100

```
mov.w #0x25A3, r12
mov.w #0x8b94, r15
bit.w r15, r12
```

Operation:
0010 0101 1010 0011 (R12) AND
1000 1011 1001 0100 (R15) =
0000 0001 1000 0000



Bitwise Logic

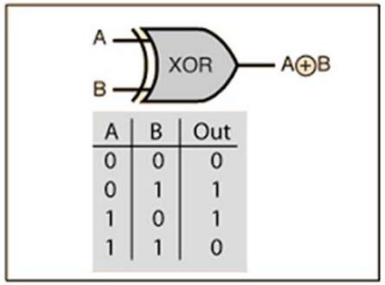
`XOR (.B or .W) src, dst;` Source and destination are XORed and the result is saved to destination.

Ex: Run the following program and observe the output.

```
mov.w #0x17E1, r15
xor.b #0x75, r15
```

OPERATION

1110 0001 (E1H)
⊕ 0111 0101 (75H)
1001 0100 (94H)



R15 | 17E1H | XOR.B 0x75 → R15 | 0094H

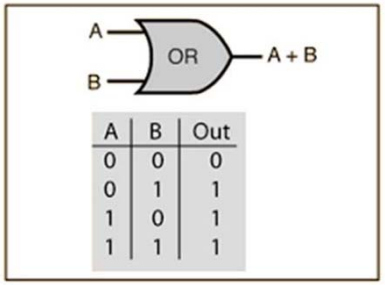
Bitwise Logic

`OR (.B or .W) src, dst;` Source and destination are ORed and the result is saved in the destination.

Ex: Run the following program and observe the output

```
mov.w #0x17E1, r15
or.b  #0x75, r15
```

OPERATION
1110 0001 (E1H)
0111 0101 (75H)
OR
1111 0101 (F5H)



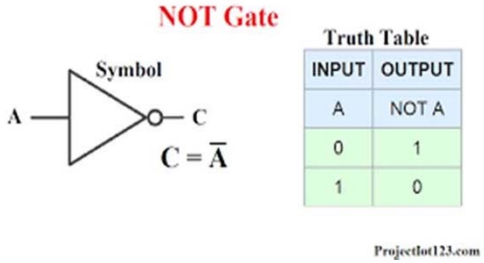
R15 | 17E1H | OR.B 0x75 → R15 | 00F5H

Bitwise Logic

`INV (.B or .W) dst;` Inverts all the bits in the destination. Result is saved in the destination.

Ex: Run the following program and observe the output

```
mov.w #0x1903, r9
inv r9
```



OPERATION

0001 1001 0000 0101 (1903H)

INV

1110 0110 1111 1010 (E6FCH)

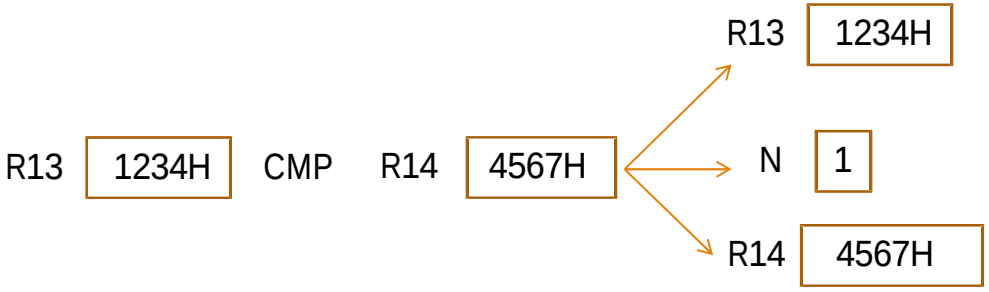
R9 | 1903H | INV R9 → R9 | E6FCH

Compare and Test

`CMP (.B or .W) src, dst;` Compare source to destination, subtracts source from destination but both are preserved. Only status register is changed.

Ex: Run the following program and observe the output.

```
mov.w #0x1234, r13
mov.w #0x4567, r14
cmp r14, r13
```



R13 and R14 are preserved but N becomes 1 since the result of subtraction is negative, other flag bits are 0.

Compare and Test

`TST (.B or .W) dst;` Test the destination for zero condition. It is very useful since **`mov`** instruction does not affect zero flag. It actually subtracts zero from the destination and check the result.

Ex: Verify the content of R12 is zero or not

```
mov.w #0x1903, r12
tst r12
```

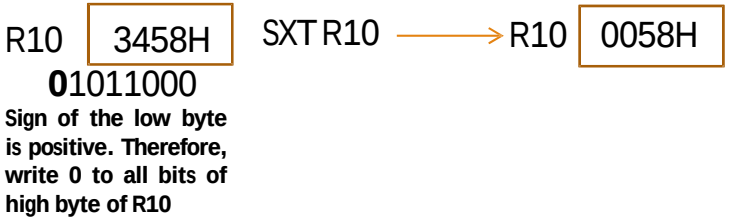
R12 is preserved and Z becomes 0 since the result of the subtraction is NOT 0!

Compare and Test

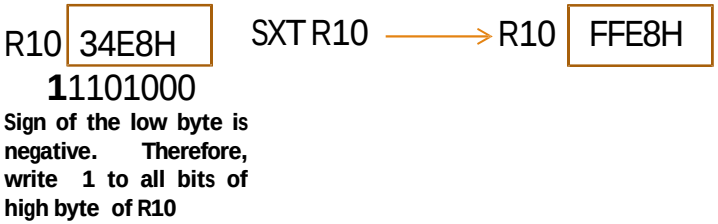
`SXT (only .W) dst;` Sign extend destination, sign of the low byte is copied to the high byte. **dst(bits 8<-->15) = dst(bit 7)**

Ex: Run following programs and observe the R10 content

```
mov.w #0x3458, r10
sxt r10
```



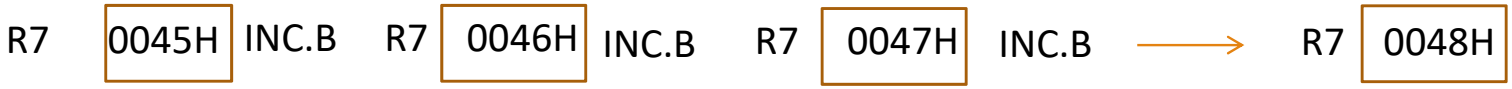

```
mov.w #0x34E8, r10
sxt r10
```



Increment

INC(.B or .W)dst; Increment destination. The destination is incremented by 1

```
Ex. mov.b #0x45, r7;copy 45H to r7 register
    inc.b r7;increase r7 content by 1, content of R7 is 46H now
    inc.b r7;increase r7 content by 1, content of R7 is 47H now
    inc.b r7;increase r7 content by 1, content of R7 is 48H now
```



Increment

INCD(.B or .W)dst; Increment destination. The destination is incremented by 2

```
Ex. mov.b #0x45, r7;copy 45H to r7 register
    incd.b r7;increase r7 content by 2, content of R7 is 47H now
    incd.b r7;increase r7 content by 2, content of R7 is 49H now
    incd.b r7;increase r7 content by 2, content of R7 is 4BH now
```

Increment

Changing addressing mode…

Ex. Write the following program and observe the output. Assume 0204H address's initial content is zero

```
mov #0x0200, r5; copy 200h to R5 register
inc.b 4(r5);increment the content of memory location indexed by R5+4
incd.b 4(r5);increment double the content of memory location indexed by R5+4
```

Increment

Changing addressing mode…

Ex.

```
mov.w #0x2345, r4; copy 2345H to R4 register
mov.w #0x5789, r5; copy 5789H to R5 register
sub r4, r5      ; subtract R4 from R5 and save the result to R5
inc.b r5        ; increase LSB of R5 by 1 and save it to R5
```

Decrement

DEC(.B or .W)dst; Decrement destination. The destination is decremented by 1.

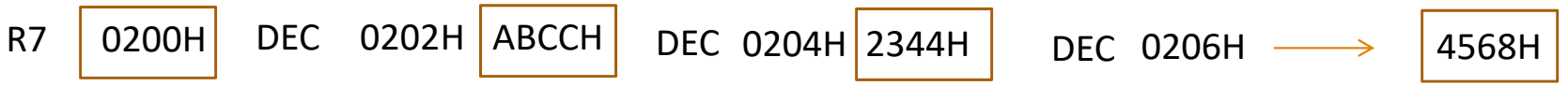
Ex:

```
mov.w #0x2345, r8; copy 2345H to R8 register
dec r8; decrease R8 content by 1, content of R8 is 2344H
dec.w r8; decrease R8 content by 1, content of R8 is 2343H
dec.b r8; decrease R8 LSB content by 1, content of R8 is 0042H
```

Decrement

Ex: Assume 0x0202h address has ABCDH, 0x0204H has 2345H and 0x0206H has 4569H initially and run the following program.

```
mov #0x0200, r7; copy 200h to R7 register
dec 2(r7); decrement 202h address content by 1, (0202h)= ABCCH
dec 4(r7); decrement 204h address content by 1, (0204h)= 2344H
dec 6(r7); decrement 204h address content by 1, (0206h)= 4568H
```



Decrement

DECD(.B or .W)dst; Decrement destination. The destination is decremented by 2.

Ex:

```
mov.w #0x6937, r9; copy 6937H to R9 register
decd r9; decrease R9 content by 2, content of R9 is 6935H
decd.w r9; decrease R9 content by 2, content of R9 is 6933H
decd.b r9; decrease R9 LB content by 2, content of R9 is 31H
```

Examples

Ex:

```
mov.w #0x234D, r15 ;R15=234DH
xor.w #0x4575, r15 ;R15=234DH XOR 4575H, R15= 6638H
inc r15     ;R15=6639H
incd r15    ;R15=663BH
incd.w r15 ;R15=663DH
dec.w r15  ;R15=663CH
decd.w r15 ;R15=663AH
decd.b r15 ;R15=0038H
```

Examples

Ex:

```
mov.w #0x2135, r12 ;R12=2135H
mov.w #0x4364, r13 ;R13=4364H
add.w r12, r13 ;R13=6499H
mov.w #0x1111, r11 ;R11=1111H
sub.w r11, r13 ;R13=5388H
xor.w #0x3245, r13 ;R13=61CDH
decd.w r13 ;R13=61CBH
incd.b r13 ;R13=00CDH
```