LECTURE # 11

SIGNED NUMBER ARITHMATIC OPERATIONS

Signed Byte operands

- In signed byte operands, D7 (MSB) is the sign and D0 to D6 are set aside for the magnitude of the number.
- If D7 = 0, the operand is positive.
- If D7 = 1, the operand is negative.

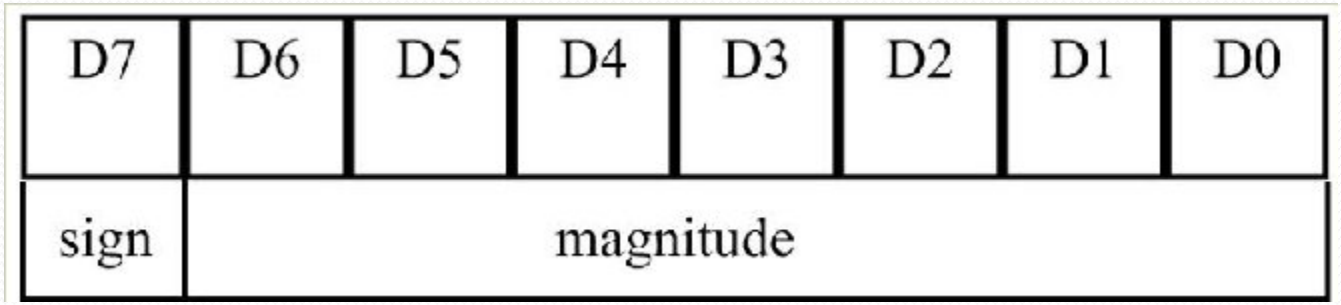


Figure 1: Signed byte operand

Signed Byte operands

Positive Numbers:

- The range of positive numbers is 0 to +127.
- If a positive number is greater than 127, a word size operand must be used.

0	0000	0000
+1	0000	0001
+5	0000	0101
. . .	. . .	. . .
+127	0111	1111

Signed Byte operands

Negative Numbers:

- For negative numbers D7 is set to 1, but the magnitude is represented in 2's complement.
- Although the assembler does the conversion, it is still important to understand how the conversion works.
- The range of byte sized negative numbers is -1 to -128.
- To convert to negative number representation (2's complement), follow these steps:
  - Write the magnitude of the number in 8 bit binary (no sign).
  - Invert each bit.
  - Add 1 to it.

Signed Byte operands

- The following lists the byte sized signed number ranges:

Decimal	Binary	Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
. . .	. . .	. . .
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
. . .	. . .	. . .
+127	0111 1111	7F

Signed Byte operands

Example: Show how the computer would represent -5.

Solution:

- 0000 0101 5 in 8-bit binary
- 1111 1010 invert each bit
- 1111 1011 add 1 (hex = FBH)

This is the signed number representation in 2's complement for -5.

Signed Byte operands

Example: Show **-**34H as it is represented internally.

Solution:

- 0011 0100
- 1100 1011
- 1100 1100 (hex = CCH)

This is the signed number representation in 2’s complement for -34H.

Signed Byte operands

Example: Show the representation for -128.

Solution:

- 1000 0000
- 0111 1111
- 1000 0000

Notice that this is negative zero (-0)

Overflow problem in signed numbers

- When using signed numbers, a serious problem arises that must be dealt with.
- This is the overflow problem. The CPU indicates the existence of problem by raising the overflow flag, but it is up to the programmer to take care of it.
- The CPU understands only 0s and 1s and ignores the human convention of positive and negative numbers.

Overflow problem in signed numbers

Example:

Look at the following code and data segments:

```
DATA1 DB      +96
DATA2 DB      +70
              . . .      . . .
      MOV     AL, DATA1      ; AL = 0110 0000        (AL=60H)
      MOV     BL, DATA2      ; BL = 0100 0110        (BL=46H)
      ADD     AL, BL         ; AL = 1010 011     (AL=A6H=90)
```

+ 96	0110 0000
+ 70	0100 0110
+166	1010 0110

According to the CPU, this is 90, which is wrong (OF=1, SF=1, CF=0)

Overflow problem in signed numbers

- In the example above, +96 is added to +70 and the result according to the CPU is -90. Why?
- The reason is that the result is more than what AL could handle.
- Like all other 8 bit registers, AL could only contain up to +127.
- The designers of the CPU created the overflow flag specifically for the purpose of informing the programmer that the result of the signed number operation has some error.

Overflow problem in signed numbers

When the overflow flag is set in 8-bit operations:

In 8-bit signed number operations, OF is set to 1 if either of the two following two conditions occur:

- There is a carry from D6 to D7 but no carry out of D7 (CF=0)
- There is a carry from D7 out (CF=1) but no carry from D6 to D7.

The overflow flag is low (not set) if there is a carry from D6 to D7 and from D7 out. The OF is set to 1 only when there is a carry from D6 to D7 or from D7 out, but not from both.

Overflow problem in signed numbers

When the overflow flag is set in 16-bit operations:

In 16-bit signed number operations, OF is set to 1 if either of the two following two conditions occur:

- There is a carry from D14 to D15 but no carry out of D15 (CF=0)
- There is a carry from D15 out (CF=1) but no carry from D14 to D15.

The overflow flag is low (not set) if there is a carry from D14 to D15 and from D15 out. The OF is set to 1 only when there is a carry from D14 to D15 or from D15 out, but not from both.

Overflow problem in signed numbers

Example: Observe the results of the following:

```
MOV  DL, -128      ; DL= 1000 0000    (DL=80H)
MOV  CH, -2        ; CH= 1111 1110    (CH=FEH)
ADD  DL, CH        ; DL= 0111 1110    (DL=7EH=+126)
```

```
-128   1000 0000
  -2   1111 1110
-130   0111 1110 (OF=1, SF=0, CF=1)
```

According to the CPU, the result is +126, which is wrong. The error is indicated by the fact that OF = 1.

Overflow problem in signed numbers

Example: Observe the results of the following:

```
MOV  AL, -2        ; AL= 1111 1110    (AL=FEH)
MOV  CL, -5        ; CL= 1111 1011    (CL=FBH)
ADD  CL, AL        ; CL= 1111 1001    (CL=F9H=7)
```

```
-2      1111 1110
-5      1111 1011
-7      1111 1001 (OF=0, SF=1 [negative], CF=1)
```

According to the CPU, the result is 7, which is correct, since OF = 0.

Overflow problem in signed numbers

Example: Observe the results of the following:

```
MOV  DH, +7        ; DH= 0000 0111    (DH=07H)
MOV  BH, +18       ; BH= 0001 0010    (BH=12H)
ADD  BH, DH        ; BH= 0001 1001    (BH=19H=+25)
```

```
+7      0000 0111
+18     0001 0010
+25     0001 1001 (OF=0, SF=0, CF=0)
```

According to the CPU, the result is +25, which is correct, since OF = 0.

Overflow problem in signed numbers

Example: Observe the results of the following:

```
MOV  AX, 6E2FH   ; 28,207
MOV  CX, 13D4H   ; 5,076
ADD  AX, CX      ; = 33,283
```

```
  6E2F        0110 1110 0010 1111
+13D4         0001 0011 1101 0100
  8203        1000 0010 0000 0011        = -32,253
```

Since OF = 1, SF = 1 (negative) and CF = 0

Overflow problem in signed numbers

Example: Observe the results of the following:

```
MOV DX, 542FH    ; 21,551
MOV BX, 12E0H    ; 4,832
ADD DX, BX       ; = 26383
```

```
  542F      0101 0100 0010 1111
+12E0       0001 0010 1110 0000
  670F      0110 0111 0000 1111      = 26,383
```

Since, OF = 0, CF = 0 and SF = 0 (positive)

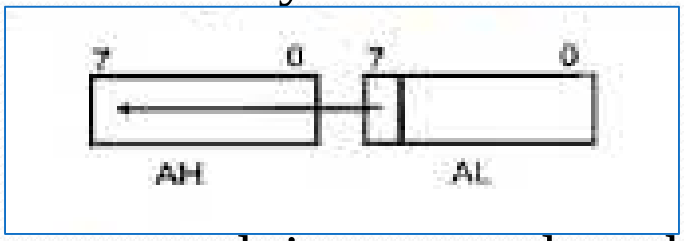
Avoiding erroneous results in signed number operations:

- To avoid the problems associated with signed number operations, one can sign extend the operand.
- Sign extension copies the sign bit (D7) of the lower byte of the register into the upper bits of the register, or copies the sign bit of a 16 bit register into another register.
- **CBW** (convert signed byte to signed word) and **CWD** (convert signed word to signed double word) are used to perform sign extension.

Avoiding erroneous results in signed number operations:

CBW (convert signed byte to signed word)

- CBW will copy D7 (the sign flag) to all bits of AH.
- This instruction converts byte AL to word AX.



- Notice that the operand is assumed to be AL and the previous contents of AH are destroyed.
- Look at the following examples:

```
MOV AL, +96          ; AL=0110 0000
CBW                  ; AH=0000 0000 & AL=0110 0000
```

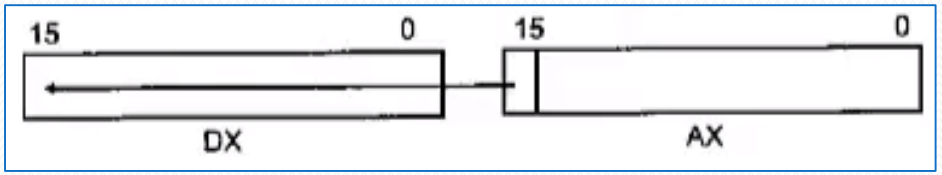
or,

```
MOV AL, -2           ; AL=1111 1110
CBW                  ; AH=1111 1111 & AL=1111 1110
```

Avoiding erroneous results in signed number operations:

CWD (convert signed word to signed double word)

- CWD sign extends AX. It copies D15 of AX to all bits of the DX register.
- This is used for signed word operands.



- Look at the following examples:

```
        MOV AX, +260        ; AX=0000 0001 0000 0100 or 0104H
        CWD                 ; DX=0000H & AX=0104H
```

or,

```
        MOV AX, -32766      ; AX=1000 0000 0000 0010 or 8002H
        CWD                 ; DX=FFFFH & AX=8002H
```

ARITHMETIC SHIFT

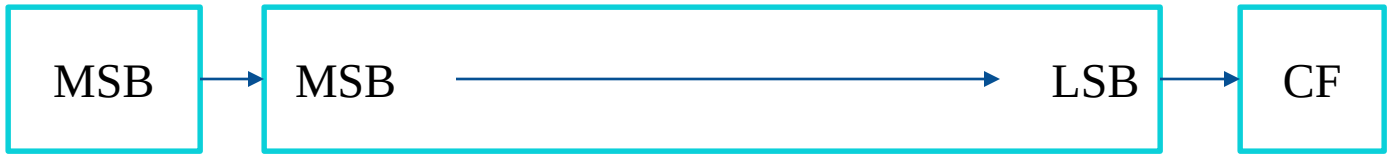
Arithmetic Shift

- As we were discussed earlier that shift instruction has two types: logical and arithmetic.
- Logical shift is used for unsigned numbers.
- The arithmetic shift is used for signed numbers.
- It is basically same as the logical shift, except that the sign bit is copied to the shifted bits.
- SAR (shift arithmetic right) and SAL (shift arithmetic left) are two instructions used for arithmetic shift.

Arithmetic Shift

SAR (shift arithmetic right)

Syntax: SAR destination, count



As the bits of the destination are shifted to the right into CF, the empty bits are filled with the sign bit.

Example:

```
MOV  AL, -10        ; AL = -10 = F6H = 1111 0110
SAR  AL,1           ; AL = 1111 1011 (CF = 0)
```

Arithmetic Shift

SAL (shift arithmetic left)

- SAL and SHL do exactly the same thing.
- It is basically the same instruction with mnemonics.
- As far as signed numbers are concerned, there is no need for SAL.

SIGNED NUMBER COMPARISON

Signed Number Comparison

Syntax:

CMP destination, source

- CMP instruction is same for both signed and unsigned numbers, the jump instruction used to make a decision for the signed numbers is different from that used for the unsigned numbers.
- In unsigned numbers comparisons CF and ZF are checked for condition of larger, equal and smaller.
- In signed numbers comparison, OF, ZF and SF are checked:

destination > source	OF = SF or ZF = 0
destination = source	ZF = 1
destination < source	OF = negation of SF

Signed Number Comparison

- The mnemonics used to detect the conditions above are as follows:

JG	Jump greater	if OF=SF or ZF=0
JGE	Jump greater or equal	if OF=SF
JL	Jump less	if OF=~SF
JLE	Jump less or equal	if OF=~SF or ZF=1
JE	Jump if equal	if ZF=1

Signed Number Comparison

Program: Find the lowest temperature as follows:

+13, -10, +19, +14, -18, -9, +12, -19, +16

```
        .MODEL SMALL
        .STACK 64
        .DATA
TEMP    DB      +13, -10, +19, +14, -18, -9, +12, -19, +16
        ORG     10H
LOWEST DB       ?
        .CODE
MAIN    PROC FAR
        MOV     AX, @DATA
        MOV     DS, AX
        MOV CX, 8
        MOV SI, OFFSET TEMP
        MOV AL, [SI]
BACK:   INC SI
        CMP AL, [SI]
        JLE SEARCH
        MOV AL, [SI]
SEARCH: LOOP BACK
        MOV LOWEST, AL
        MOV AH, 4CH
        INT 21H
MAIN    ENDP
        END MAIN
```