EEE 410 – Microprocessors I

Fall 05/06 – Lecture Notes # 13

Outline of the Lecture

- **Signed Numbers and Signed Number Operations**

SIGNED NUMBER ARITHMETIC OPERATIONS

- Until now we have seen unsigned numbers where entire 8-bit or 16-bit operand was used for the magnitude.
- In order to represent positive and negative numbers signed numbers have been introduced. The representation of signed numbers:
  - The MSB is set aside for the **sign** (+ or –) and the rest of the bits are used for the magnitude.
  - The sign is represented by 0 for positive (+) numbers and 1 for (–) negative numbers.

Signed byte operands:

D7	D6	D5	D4	D3	D2	D1	D0
sign							

If D7=0 the operand is positive
If D7=1 it is negative.

Positive Numbers:

The range of positive numbers that can be represented as a signed byte operand is **0 to +127.**

Ex:		
	0	0000 0000
	+1	0000 0001
	+5	0000 0101
	::	::::::::::::::
	+127	0111 1111

Note: If a positive number is larger than +127, a word-size operand must be used.

Negative Numbers:

For negative signed numbers D7=1, but the magnitude operand is represented in 2's complement. Although the assembler does the conversion, it is important to understand how the conversion works.

To convert to negative number representation (2's complement) follow the steps:

1. Write the magnitude of the number in 8-bit binary (no sign)
2. Invert each bit
3. Add 1 to it

Ex: Show how the computer would represent –5

1.	0000 0101	5 in 8-bit binary
2.	1111 1010	invert each bit
3.	1111 1011	add 1 (hex = FBH)

This is the signed number representation of –5 in 2's complement.

Ex: Show how the computer would represent –52

1.	0011 0100	
2.	1100 1011	
3.	1100 1100	(CCH)

Ex: Show the representation of –128

1.	1000 0000	
2.	0111 1111	
3.	1000 0000	(80H) Notice this is not negative zero (–0)

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

Word-sized byte operands:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

sign

If D15=0 the operand is positive
If D15=1 it is negative.

Can be used for the representation of numbers between –32768 to +32767. Larger numbers must be treated as a multiword numbers as unsigned numbers.

Decimal	Binary	Hex
–32 768	1000 0000 0000 0000	8000
–32 767	1000 0000 0000 0001	8001
–32 766	1000 0000 0000 0010	8002
: :	: : : : : : : :	: :
–2	1111 1111 1111 1110	FFFE
–1	1111 1111 1111 1111	FFFF
0	0000 0000 0000 0000	0000
+1	0000 0000 0000 0001	0001
+2	0000 0000 0000 0010	0002
: :	: : : : : : : :	: :
+32 766	0111 1111 1111 1110	7FFE
+32 767	0111 1111 1111 1111	7FFF

Overflow problem in signed number operations

When using signed numbers Overflow problem can arise after an operation. This problem arises if the result in a register after an operation is too large. In such a case CPU sets the OF (Overflow Flag). The programmer must consider the overflow case.

```
Ex:   DATA1   DB   +96
      DATA2   DB   +70
              …    …
              MOV  AL,DATA1      ;AL=0110 0000 (60H)
              MOV  BL,DATA2      ;BL=0100 0110 (46H)
              ADD  AL,BL         ;AL=1010 0110 (AL=A6H=-90  invalid!)
```

```
+ 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)
```

As defined before max positive signed number for an 8-bit register is +127. Because +166 is greater than +127 the problem is arising. The overflow flag is set to inform the programmer that there is erroneous result from the signed number operation above.

When the OF is set in 8-bit operations

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

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

```
Ex:             MOV  DL,–128        ;DL=1000 0000 (80H)
                MOV  CH, –2         ;CH=1111 1110 (FEH)
                ADD  DL,CH          ;DL=0111 1110 (DL=FEH=+126  invalid!)

-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.

```
Ex:             MOV  AL,–2          ;AL=1111 1110 (FEH)
                MOV  CL,–5          ;CL=1111 1011 (FBH)
                ADD  CL,AL          ;CL=1111 1001 (CL=F9H=-7  which is correct!)

  -2   1111  1110
+ -5   1111  1011
  -7   1111  1001        OF=0, SF=1 (negatieve) , CF=1   : The result is correct since OF=0.
```

```
Ex:             MOV  DH,+7          ;DH=0000 0111 (FEH)
                MOV  BH,+18         ;BH=0001 0010 (FBH)
                ADD  BH,DH          ;BH=0001 1001 (CL=19H=+25  which is correct!)

   +7  0000  0111
+ +18  0001  0010
  +25  0001  1001        OF=0, SF=0 (positive) , CF=0   : The result is correct since OF=0.
```

OF in 16-bit operations

In 16-bit signed number operations, OF is set to 1 in either of the cases:

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

```
Ex:             MOV  AX,62FH        ;28 207        (MOV  AX,+28807))
                MOV  CX,13D4H       ;  5076
                ADD  AX,CX          ;=33283 is expected result (out of range)
  6E2F       0110  1110 0010 1111
+ 13D4       0001  0011 1101 0100
  8203       1000  0010 0000 0011 = –32,253   incorrect!      OF=1, SF=1, CF=0
```

Avoiding erroneous results in signed number operations

- In order to avoid the problem of signed number operations we can sign extend the operand. Sign extension copies the sign bit (D7) of the lower byte of a register to the upper byte bits of of the register, or copies the sign bit of a 16-bit register into another register.
- There are two commands used for sign extension.

CBW ; Convert signed Byte to signed Word

CBW will copy D7 (the sign flag) of **AL** to all bits of **AH**. Notice that the operand is assumed to be AL and the contents of AH is destroyed.

```
Ex:   MOV AL,+96      ;AL = 0110 0000
      CBW             ;now AH= 0000 0000 and AL=0110 0000
```

```
Ex:   MOV AL,–2       ;AL = 1111 1110
      CBW             ;now AH= 1111 1111 and AL=1111 1110
```

CWD ; Convert signed Word to signed Doubleword

CWD will copy D15 (the sign flag) of **AX** to all bits of **DX**. Notice that the operand is assumed to be AX and the contents of DX is destroyed.

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

```
Ex:   MOV AX,–32766   ;AX = 1000 0000 0000 0010B or AX=8002H
      CWD             ;DX = FFFFH and AX=8002H
```

- How can these instructions help correct the overflow error?

Lets give an example program which takes into consideration of correction of signed byte addition operation.

```
Ex:   DATA1    DB    +96
      DATA2    DB    +70
      RESULT   DW    ?
               …
               MOV   AH,0          ;AH=0
               MOV   AL,DATA1      ;get operand 1
               MOV   BL,DATA2      ;get operand 2
               ADD   AL,BL         ;add them
               JNO   OVER          ;jump if there is no overflow (OF=0) to OVER
               MOV   AL,DATA2      ;otherwise get operand 2 to
               CBW                 ;sign extend it
               MOV   BX,AX
               MOV   AL,DATA1      ; get back operand 1 to
               CBW                 ;sign extend it
               ADD   AX,BX         ;add them
      OVER:    MOV   RESULT,AX     ;save the result
```