

Lecture 9

- Arithmetic Operations
- Overflow
- Multiply and Divide

INC and DEC

- INC – adds one to a single operand
- DEC – decrements one from a single operand

INC/DEC Examples

```
inc  al    ; increment 8-bit register
dec  bx    ; decrement 16-bit register
inc  membyte ;increment 8-bit
                ;memory operand
dec  membyte ; decrement 8-bit
                ;memory operand
inc  memword ;increment 16-bit
                ;memory operand
dec  memword ;decrement 16-bit
                ;memory operand
```

ADD

- ADD adds a source operand to a destination operand *of the same size*.

```
ADD dest, src ;dest = src+dest
```

ADD Examples

```
add cl, al      ;add 8-bit register to register
                ;cl = cl + al
add bx, 1000h  ;add immediate to 16-bit
                ; register – bx = bx + 1000h
add var1, ax   ;add 16-bit register to memory
                ;var1 = var1 + ax
add dx, var1   ;add 16-bit memory to register
                ;dx = dx + var1
add var, 10    ;add immediate value to
                ;memory – var = var + 10
```

SUB

- SUB subtracts a source operand from a destination operand.
SUB dest, src ;dest = dest – src

SUB Examples

```
sub cl, al      ;subtract 8-bit
                ; register from register
                ;cl = cl - al
sub bx, 1000h   ;subtract immediate
                ;value from 16-bit
                ;register
                ; bx = bx - 1000h
sub var1, ax    ;subtract 16-bit register
                ; from memory
                ;var1 = var1 - ax
sub dx, var1    ;subtract 16-bit
                ;memory from register
                ;dx = dx – var1
sub var1, 10    ;subtract immediate
                ;value from memory
                ;var1 = var1 - 10
```

Flags Affected: Zero

- Zero flag – set when the result of an operation is zero.
mov ax, 10
sub ax, 10 ;ax = 0, ZF = 1
mov bl, 4Fh
add bl, 0B1h ;bl = 00, ZF = 1
mov ax, 0FFFFh ;ax = -1
inc ax ;ZF = 1
mov ax, 1
dec ax ;ZF = 1

Flags Affected: Sign

- Sign flag – set when the result of an operation is negative.

```
mov bx, 1  
sub bx, 2    ;bx = FFFF, SF = 1
```

Flags Affected: Carry

- Carry flag – set when there is a carry out of the left-most bit.

```
mov bl, 4Fh  
add bl, 0B1h ;bl = 00, CF = 1
```

Flags Affected: Overflow

- Overflow flag – set when an arithmetic operation generates a signed value that exceeds to storage size of the destination operand.

Overflow

- Checking for Overflow

Unsigned Overflow

- During unsigned arithmetic, if the carry flag is set then overflow has occurred:

More on Unsigned Overflow

- Something looks strange about the subtraction:
 $01 + FE = FF \rightarrow$ where is the carry?
- For subtraction and negation, the carry flag is set if there is NO carry out of the most significant bit.
 $1 - 2 = 01 + FE = FF$ – no carry, so carry bit is set to signify overflow.
 $2 - 1 = 02 + FF = 01$ – there is a carry, and the carry bit is cleared, signifying no overflow.
- Yes, this is a bit counter -intuitive.

Signed Overflow

- During signed arithmetic, the overflow flag is set when an out of range value is generated.

Signed Overflow Examples

```
mov al, 126  
add al, 2
```

```
01111110  
+ 00000010  
10000000 = 80h  
carry into the sign bit is 1  
carry out of sign bit is 0  
80h = -128, not +128 -> overflow
```

More Signed Overflow

```

mov al, -128
sub al, 2

10000000
- 00000010
01111110  al = 7Eh = 126, not -130

10000000
+ 11111110 (2 in 2's comp)
1 01111110
carry in = 0, carry out = 1 -> overflow

```

Multiplication and Division

- Instructions for integer multiplication on 8, 16, and 32 bit operands
- MUL, DIV –
- IMUL, IDIV –

DIV

- DIV divides unsigned 8-bit, 16-bit, and 32-bit numbers.
- Format:
DIV *divisor*
divisor - register or memory (not immediate!)
- Registers used:

Dividend	Divisor	Quotient	Remainder
AX	op-8	AL	AH
DX:AX	op-16	AX	DX

DIV Examples

```

bval db 2 ;divisor
...
mov ax, 0083h ;dividend
div bval

```

Before DIV:

AH	AL
00	83h

After DIV:

AH	AL
01	41h

83h / 02h = 41h, R=1

DIV Examples

```
wval dw 100h ;divisor
...
mov dx, 0 ;clear dividend high!
mov ax, 8003h ;dividend
div wval
```

Before DIV:

DX	AX
0000	8003h

After DIV:

DX	AX
0003	0080h

$8003h / 0100h = 80h, R=3$

Warning

- If you don't remember to clear DX you will get unexpected results!!!
- If you had something you were using in DX, it will get destroyed by the division.

IDIV

- IDIV works like DIV except it uses signed numbers.
- In 8-bit division, the dividend is in AX, so the sign is determined by bit 15.
- In 16-bit division, the sign is determined by bit 15 in DX.

IDIV Examples

```
bval db 5 ;divisor
...
mov ax, -48 ;dividend
idiv bval
```

Before IDIV:

AH	AL
FF	D0

After DIV:

AH	AL
FD	F7

$FFD0/5 = F7, R FD$
 $F7 = -9, FD = -3$

What NOT to Do!

```
bval db 5 ;divisor
...
mov ah, 0
mov al, -48 ;dividend
idiv bval
```

Before IDIV:

AH	AL	
00	D0	(+208)

After DIV:

AH	AL
03	29

00D0/5 = 29, R 3
29 = 41

Sign Extending

- You will need to sign extend your dividend.
- Intel provides instructions for this:
 - CBW – convert byte to word extends AL into AX
 - CWD – convert word to double word extends AX into DX:AX

IDIV Examples

```
wval dw 256 ;divisor
...
mov ax, -5000 ;DX:AX = ???EC78h
cwd ;DX:AX = FFFFEC78h
idiv wval
```

Before IDIV:

DX	AX
FFFF	EC78

After DIV:

DX	AX
FF78	FFED

FFED = -19 (quotient)
FF78 = -136 (remainder)

Division Problems

- Divide Overflow is produced by:
- You'll need to prevent it:

MUL

- Multiplies an 8, 16, or 32 bit operand by AL, AX, or EAX respectively.
- Format:
 MUL multiplier
 multiplier – register or memory (not immediate!)
- Registers used:

Multiplicand	Multiplier	Product
AL	op-8	AX
AX	op-16	DX:AX

MUL Examples

```
bval db 10
...
mov al, 100
mul bval
```

Before MUL:

AH	AL
?	64h

After MUL:

AH	AL
03	E8h

MUL Examples

```
wval dw 1000
...
mov ax, 5555 ;D903h
mul wval
```

Before MUL:

DX	AX
?	D903

After MUL:

DX	AX
034F	B3B8

- Multiplication result might need a high-order byte or word.

Example: Checking Size

```
; multiply CX by AX. If result extends
; into DX, copy 2 words to result
; locations, else copy one word.
.data
ResultLo  dw ?
ResultHi  dw ?
...
mul       cx
mov       ResultLo, AX
jnc       L1 ;jump if carry not set
mov       ResultHi, DX
L1:
```

IMUL

- IMUL multiplies *signed* binary numbers.
- Why is this different?

IMUL Examples

```
bval db 4
...
mov al, -4
imul bval
```

Before IMUL:

AH	AL
?	FC

After IMUL:

AH	AL
FF	F0

CF = 0, OF = 0 : Result fits into 8 bits, AH holds sign extension.

IMUL Examples

```
bval db 4
...
mov al, 48
imul bval
```

Before IMUL:

AH	AL
?	30h

After IMUL:

AH	AL
00	C0

CF = 1, OF = 1 : Result does not fit into 8 bits!
(why? C0 is a negative number! 00C0 is positive)

IMUL Examples

```
wval dw 4
...
mov ax, 48
imul wval
```

Before IMUL:

DX	AX
?	0030

After IMUL:

DX	AX
0000	00C0

CF = 0, OF = 0: Result fits into 16 bits.

Warning

- Sixteen bit multiplication will wipe out whatever is in **DX**!
- It's easy to forget this if you are only using the result returned in **AX**.