# Computer Org. & Assembly Language Lab

## Lab#09: Integer Arithmetic - I

### Agenda

- .IF Directive
- .REPEAT Directive
- .WHILE Driective
- Shift and Rotate Instructions
    - SHL Instruction
    - SHR Instruction
    - SAL and SAR Instructions
    - ROL and ROR Instructions
    - RCL and RCR Instructions

## .IF Directive

The Microsoft assembl er (MASM) provides .IF, ahigh-leveldirective that makes programming compound IF statements much easier than if you were to code using CMP and conditional jump instructions. Here is the syntax:

```
.IF condition 1

      statements

[.ELSEIF condition2

      statements ]

[.ELSE

      statements ]

.ENDIF
```

The square brackets above show that .ELSEIF and .ELSE are optional, whereas .IF and .ENDIF are required. A condition is a Boolean expression involving the same operators used in C++ and Java (such as <, >, =, and!=). The expression is evaluated at runtime.

| **Operator** | **Description** |
|---|---|
| expr 1= expr2 | Returns true when expr 1 is equal to expr 2. |
| expr 1!=expr2 | Returns true when expr 1 is not equal to expr 2. |
| expr 1>expr2 | Returns true whenexpr 1 is greater than expr 2. |
| expr 1>=expr2 | Returns true when expr 1 is greater than or equal to expr 2. |
| expr 1<expr2 | Returns true when expr 1 is less than expr 2. |
| expr l <=expr2 | Returns true when expr 1 is less than or equal to expr 2. |
| !expr | Returns true when expr is false. |
| expr 1&&expr2 | Performslogical AND between expr 1 and expr2. |
| expr1 \|\| expr2 | Performs logical OR between expr 1 and expr2. |
| expr1&expr2 | Performs bitwise AND between expr 1 and expr 2. |
| CARRY? | Returns true if the Carry flag is set. |
| OVERFLOW? | Returns true if the Overflow flag is set. |
| PARITY? | Returns true if the Parity flag is set. |
| SIGN? | Returns true if the Sign flag is set. |
| ZERO? | Returns true if the Zero flag is set. |

```
.data
      val1 DWORD 5
      result DWORD ?
.code
      mov eax,6
      .IF eax > val1
            mov result,1
      .ENDIF
```

## .REPEAT Directive

The .REPEAT directive executes the loop body before testing the runtimeconditionfollowing the .UNTIL directive:

```
.REPEAT
      statements
.UNTIL condition
```

## .WHILE Directive

The .WHILE directive tests the condition before executing the loop:

```
.WHILE condition
      statements
.ENDW
```

**Examples**

The following statements display the values 1through 10 using the .WHILE directive:

```
mov eax,0
.WHILE eax < 10
      inc eax
      call WriteDec
      callCrlf
```

```
.ENDW
```

The followingstatementsdisplay the values 1 through 10 using the .REPEAT directive:

```
moveax,0

.REPEAT

      inc eax

      callWriteDec

      call Crlf

.UNTIL eax == 10
```

## Shift and Rotate Instructions

Shifting means to move bits right and left inside an operand. All of the following instructions affect the Overflow and Carry flags:

```
SHL   Shift left

SHR   Shift right

SAL   Shift arithmetic left

SAR   Shift arithmetic right

ROL   Rotate left

ROR   Rotate right

RCL   Rotate carry left

RCR   Rotate carry right

SHLD  Double-precision shift left

SHRD  Double-precision shift right
```

**While SHL/SHR do an unsigned rotation, SAL/SAR do a signed rotation.**

### SHL Instruction

The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0. The highest bit is moved to the Carry flag, and the bit that was in the Carry flag is lost.

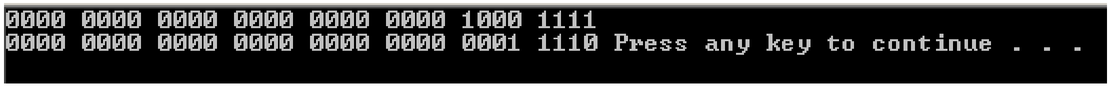
```
SHL destination, count
```

```
include irvine32.inc

.data

.code
main proc
      xor eax,eax
      mov al, 10001111b
      call writebin
      call crlf
      shl al, 1
      call writebin
exit
main endp
end main
```

**Output**

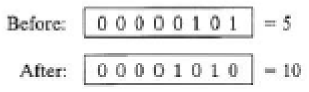


```
0000 0000 0000 0000 0000 0000 1000 1111
0000 0000 0000 0000 0000 0000 0001 1110 Press any key to continue . . .
```

**Fast Multiplication**: One of the best uses of SHL is for performing high-speed multiplication by powers of 2. Shifting any operand left by n bits multiplies the operand by $2^n$. For example, shifting 5 left 1 bit yields the product 5 * 2:

```
mov dl,5

shl dl,1
```



## SHR Instruction

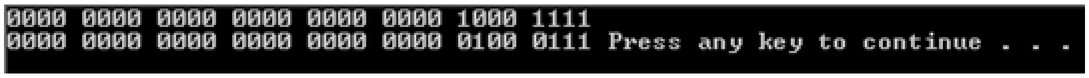
The SHR (shift right) instruction performs a logical right shift on the destination operand, replacing the highest bit with a 0. The lowest bit is copied into the Carry flag, and the bit that was in the Carry flag is lost.

```
include irvine32.inc

.data

.code
main proc
      xor eax,eax
      mov al, 10001111b
      call writebin
      call crlf
      shr al, 1
      call writebin
exit
main endp
end main
```

**Output**

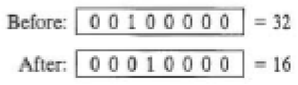


```
0000 0000 0000 0000 0000 0000 1000 1111
0000 0000 0000 0000 0000 0000 0100 0111 Press any key to continue . . .
```

**Fast Division:** Logically shifting any unsigned operand right by n bits divides the operand by $2^n$. Here, for example, we divide 32 by $2^1$, producing 16:

```
mov dl, 32

shr dl,1
```



```
Before: 0 0 1 0 0 0 0 0  = 32
After:  0 0 0 1 0 0 0 0  = 16
```

## SAL and SAR Instructions

SAL (shift arithmetic left) is identical to the SHL instruction . The SAR (shift arithmetic right) instruction performs a right arithmetic shift on its destination operand.

The syntax and operands for SAL and SAR are identical to those for SHL and SHR . The shift may be repeated, based on the counter in the second operand:

```
SAR destination, count
```

```
mov al,0F0h ;AL = 11110000b (-16)
sar al,1    ;AL = 11111000b (-8) CF = 0
```

## ROL and ROR Instruction

The ROL (rotate left) instruction shifts each bit to the left. Also, the highest bit is copied both into the Carry flag and into the lowest bit.

The ROR (rotate right) instruction shifts each bit to the right. Also, the lowest bit is copied into the Carry flag and into the highest bit at the same time.

The instruction format is the same as for the SHL instruction.

```
mov al,40h  ;AL 01000000b
rol al,1    ;AL 10000000b, CF 0
rol al,1    ;AL 00000001b, CF 1
rol al,1    ;AL 00000010b, CF 0
```

You can use ROL to exchange the upper (bits 4-7) and lower (bits 0-3) halves of a byte. For example, if 26h is rotated four bits in either direction, it becomes 62h:

```
mov al,26h
rol al,4 ; AL = 62h
```

In the following example, the lowest bit is copied into the Carry flag and into the highest bit of the result:

```
mov al,Olh ;AL OOOOOOOlb
ror al,l ;AL lOOOOOOOb, CF 1
ror al,l ;AL OlOOOOOOb, CF 0
```

## RCL and RCR Instructions

The RCL (rotate carry left) instruction shifts each bit to the left, copies the Carry flag to the least significant bit (LSB), and copies the most significant bit (MSB) into the Carry flag

```
clc                          ;CF = 0
mov bl,88h                   ;CF = 0 , BL = 10001000b
rcl bl,l                     ;CF = 1 , BL = 00010000b
rcl bl,l                     ;CF = 0 , BL = 00100001b
```

The RCR (rotate carry right) instruction shifts each bit to the right , copies the Carry flag into the most significant bit, and copies the least significant bit into the Carry flag.

In the following example, STC sets the Carry flag before rotating the Carry flag into the MSB,and rotating the LSB into the Carry flag:

```
stc                          ;CF = 1
mov ah,10h                   ;AH = 00010000,CF = 1
rcr ah,1                     ;AH = 10001000 ,CF =  0
```