Lab 10

tooba.ali@nu.edu.pk

Library Procedures – Overview

o CloseFile: Closes an open disk file

o Clrscr: Clears console, locates cursor at upper left corner

o CreateOutputFile: Creates new disk file for writing in output mode

o Crlf : Writes end of line sequence to standard output

o Delay: Pauses program execution for n millisecond interval

o DumpMem: Writes block of memory to standard output in hex

o DumpRegs: Displays general-purpose registers and flags (hex)

o GetCommandtail: Copies command-line args into array of bytes

o GetMaxXY: Gets number of cols, rows in console window buffer

o GetMseconds: Returns milliseconds elapsed since midnight

o GetTextColor: Returns active foreground and background text colors in the console window

o Gotoxy: Locates cursor at row and column on the console

o IsDigit: Sets Zero flag if AL contains ASCII code for decimal digit (0–9)

o MsgBox: Display popup message boxes

o MsgBoxAsk: Display a yes/no question in a popup message box

o OpenInputFile: Opens existing file for input

o ParseDecimal32: Converts unsigned integer string to binary

o ParseInteger32: Converts signed integer string to binary

o Random32: Generates 32-bit pseudorandom integer in the range 0 to FFFFFFFFh

o Randomize: Seeds the random number generator

o RandomRange: Generates a pseudorandom integer within a specified range

o ReadChar: Reads a single character from standard input

o ReadFromFile: Reads input disk file into buffer

o ReadDec: Reads 32-bit unsigned decimal integer from keyboard

o ReadHex: Reads 32-bit hexadecimal integer from keyboard

o ReadInt: Reads 32-bit signed decimal integer from keyboard

o ReadKey: Reads character from keyboard input buffer

o ReadString: Reads string from standard input, terminated by [Enter]

o SetTextColor: Sets foreground and background colors of all subsequent console text output

o StrLength: Returns length of a string

o WaitMsg: Displays message, waits for Enter key to be pressed

o WriteBin: Writes unsigned 32-bit integer in ASCII binary format.

o WriteBinB: Writes binary integer in byte, word, or doubleword format

o WriteChar: Writes a single character to standard output

o WriteDec: Writes unsigned 32-bit integer in decimal format

o WriteHex: Writes an unsigned 32-bit integer in hexadecimal format

o WriteHexB: Writes byte, word, or doubleword in hexadecimal format

o WriteInt: Writes signed 32-bit integer in decimal format

o WriteString: Writes null-terminated string to console window

o WriteToFile: Writes buffer to output file

o WriteWindowsMsg: Displays most recent error message generated by MS-Windows

For more info: visit this link-> <http://programming.msjc.edu/asm/help/index.html?page=source%2Firvinelib%2Fwritedec.htm>

# • Example

Clear the screen, delay the program for 500 milliseconds, and dump the registers and flags

.code

call Clrscr

mov eax,500

call Delay

call DumpRegs

# • Example

Display a null-terminated string and move the cursor to the beginning of the next screen line

.data

str1 BYTE "Assembly language is easy!",0

.code

mov edx,OFFSET str1

call WriteString

call Crlf

# • Example

Display a null-terminated string and move the cursor to the beginning of the next screen

line (use embedded CR/LF)

.data

str1 BYTE "Assembly language is easy!",0Dh,0Ah,0

.code

mov edx,OFFSET str1

call WriteString

# • Example

Display an unsigned integer in binary, decimal, and hexadecimal, each on a separate line IntVal = 35

.code

mov eax,IntVal

call WriteBin ; display binary

call Crlf

call WriteDec ; display decimal

call Crlf

call WriteHex ; display hexadecimal

call Crlf

# • Example 4 - Input a string from the user

EDX points to the string

ECX specifies the maximum number of characters the user is permitted to enter

.data

fileName BYTE 80 DUP(0)

.code

mov edx,OFFSET fileName

mov ecx,SIZEOF fileName – 1

call ReadString

Note: A null byte is automatically appended to the string

# • Example 5

Generate and display ten pseudorandom signed integers in the range 0 – 99 then pass each integer to WriteInt in EAX and display it on a separate line

.code

mov ecx,10 ; loop counter

L1: mov eax,100; ceiling value

call RandomRange ; generate random int

call WriteInt ; display signed int

call Crlf ; goto next display line

loop L1 ; repeat loop

# • Example 6

Display a null-terminated string with yellow characters on a blue background

.data

str1 BYTE "Color output is easy!",0

.code

mov eax,yellow + (blue \* 16)

call SetTextColor

mov edx,OFFSET str1

call WriteString

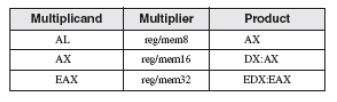
call Crlf

Note: The background color is multiplied by 16 before being added to the foreground color

Multiplication and Division Instruction

# MUL Instructions:

The **MUL** instruction is for unsigned multiplication. Operands are treated as unsigned numbers.



**Syntax:** *MUL source*

## EXAMPLE # 01

*mov al,5h*

*mov bl,10h*

*mul bl ; AX = 0050h, CF = 0*

## EXAMPLE # 02

*.data*

*val1 WORD 2000h*

*val2 WORD 0100h*

*.code*

*mov ax,val1 ; AX = 2000h*

*mul val2 ; DX:AX = 00200000h, CF = 1*

## EXAMPLE # 03

*mov eax,12345h*

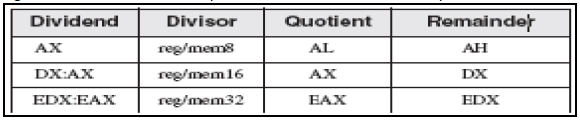
*mov ebx,1000h*

*mul ebx ; EDX:EAX = 0000000012345000h, CF = 0*

# DIV Instructions

The DIV (unsigned divide) instruction performs 8-bit, 16-bit, and 32-bit unsigned integer division. The single register or memory operand is the divisor.

The following table shows the relationship between the dividend, divisor, quotient, and remainder:



**Syntax:**

*DIV source*

*mov ax,0083h ; dividend*

*mov bl,2 ; divisor*

*div bl ; AL = 41h, AH = 01h*

*mov dx,0 ; clear dividend, high*

*mov ax,8003h ; dividend, low*

*mov cx,100h ; divisor*

*div cx ; AX = 0080h, DX = 0003h*

# Sign Extension Instructions (CBW,CWD,CDQ)

Dividends of signed integer division instructions must often be sign-extended before the division takes place. Intel provides three useful sign extension instructions: CBW, CWD, and CDQ. The CBW instruction (convert byte to word) extends the sign bit of AL into AH, preserving the number’s sign. In the next example, 9Bh (in AL) and FF9Bh (in AX) both equal −101 decimal:

## EXAMPLE # 01

.data

byteVal SBYTE -101 ; 9Bh

.code

mov al,byteVal ; AL = 9Bh

cbw ; AX = FF9Bh

The CWD (convert word to doubleword) instruction extends the sign bit of AX into DX:

.data

wordVal SWORD -101 ; FF9Bh

.code

mov ax,wordVal ; AX = FF9Bh

cwd ; DX:AX = FFFFFF9Bh

The CDQ (convert doubleword to quadword) instruction extends the sign bit of EAX into EDX:

.data

dwordVal SDWORD -101 ; FFFFFF9Bh

.code

mov eax,dwordVal

cdq ; EDX:EAX = FFFFFFFFFFFFFF9Bh

# IDIV Instructions:

**Syntax: IDIV *source***

## EXAMPLE # 01:

.data

byteVal SBYTE -48 ; D0 hexadecimal

.code

mov al,byteVal ; lower half of dividend

cbw ; extend AL into AH

mov bl,+5 ; divisor

idiv bl ;AL = -9, AH = -3

# IMUL Instructions:

The **IMUL** instruction is for signed multiplication. Operands are treated as signed numbers and result is positive or negative depending on the signs of the operands.

**Syntax: I***MUL source*

The following instructions multiply 48 by 4, producing -192 in AX. Although the product is correct, AH is not a sign extension of AL, so the Overflow flag is set:

*mov al,48*

*mov bl,4*

*imul bl ;AX = 00C0h, OF = 1*

The following instructions multiply -4 by 4, producing -16 in AX. AH is a sign extension of AL so the Overflow flag is clear:

*mov al,-4*

*mov bl,4*

*imul bl ; AX = FFF0h, OF = 0*

The following instructions perform 32-bit signed multiplication (4,823,424 \* -423), producing -2,040,308,352 in EDX:EAX. The Overflow flag is clear because EDX is a sign extension of EAX:

*mov eax,+4823424*

*mov ebx,-423*

*imul ebx ; EDX:EAX = FFFFFFFF86635D80h, OF = 0*

The following instructions demonstrate two-operand formats:

## EXAMPLE # 01

.data

word1 SWORD 4

dword1 SDWORD 4

.code

mov ax,-16 ; AX = -16

mov bx,2 ; BX = 2

imul bx,ax ; BX = -32

imul bx,2 ; BX = -64

imul bx,word1 ; BX = -256

mov eax,-16 ; EAX = -16

mov ebx,2 ; EBX = 2

imul ebx,eax ; EBX = -32

imul ebx,2 ; EBX = -64

imul ebx,dword1 ; EBX = -256

The following instructions demonstrate three-operand formats, including an example of signed overflow:

## EXAMPLE # 02

.data

word1 SWORD 4

dword1 SDWORD 4

.code

imul bx,word1,-16 ; BX = -64

imul ebx,dword1,-16 ; EBX = -64

imul ebx,dword1,-2000000000 ; OF = 1

# Exercises:

**Q1# Write an x86 assembly procedure FACTORIAL that will compute factorial (N!) for an unsigned variable N.**

**HINT:       N!** = 1 if N == 1

**N!** = N x (N – 1) x (N – 2) x … x 1 if N > 1    

**Q2# The greatest common divisor (GCD) of two integers is the largest integer that will evenly divide both integers. Implement this procedure in x86 assembly language**