CS271 Irvine Library Procedures List

CloseFile....2
ClrScr....2
CreateOutputFile....3
CrLf....3
Delay....4
DumpMem....4
DumpRegs....5
GetCommandTail....5
GetDateTime....6
GetMaxXY....7
GetMseconds....7
GetTextColor....8
GotoXY....8
IsDigit....9
MsgBox....9
MsgBoxAsk....10
OpenInputFile....11
ParseDecimal32....11
ParseInteger32....12
Random32....12
Randomize....13
RandomRange....13
ReadChar....14
ReadDec....14
ReadFloat....15
ReadFromFile....15
ReadHex....16
ReadInt....16
ReadKey....17
ReadString....18
SetTextColor....19
ShowFPUStack....19
Str_Compare....20
Str_Copy....21
Str_Length....21
Str_Trim....22
Str_UCase....22
WaitMsg....23
WriteBin....23
WriteBinB....24
WriteChar....24
WriteDec....25
WriteFloat....25
WriteHex....26
WriteHexB....26
WriteInt....27
WriteString....27
WriteToFile....28
WriteWindowsMsg....28

Oregon State University
College of Engineering

CloseFile

Close a disk file that is currently “open for editing”.

Receives:

EAX = Operating System File Handle

Returns:

EAX = Return Code (0 if there was an error in execution)

Example Code:

```
MOV  EAX, fileHandle    ; Put OS File Handle into EAX
CALL CloseFile
```

ClrScr

Clears the terminal window.

Receives:

None

Returns:

None

Example Code:

```
CALL WaitMsg      ; "Press any key..."
CALL ClrScr       ; Clear console window
```

CreateOutputFile

Creates a new disk file for writing in output mode.

Receives:

EDX = Address/filename of file to be created. Defaults to current working directory.

Returns:

EAX = OS file handle (INVALID_HANDLE_VALUE if error)

Example Code:

```
.data
 fileName    BYTE     "New_Filename.txt", 0

.code
 MOV  EDX, OFFSET fileName
 CALL CreateOutputFile
 CMP  EAX, INVALID_HANDLE_VALUE ; Check for error in file creation
 JE   _fileError ; If equal, there was an error... jump to handler
```

CrLf

Writes an end-of-line sequence (Carriage Return, Line Feed) to the terminal.

Receives:

None

Returns:

None

Example Code:

```
CALL CrLf     ; Like hitting "Enter" at the end of a line
```

Delay

Pauses program execution for a number of milliseconds. (1000ms = 1 second)

Receives:

EAX = number of milliseconds to wait before continuing execution

Returns:

None

Example Code:

```
MOV  EAX, 1000        ; 1 second
CALL Delay
```

DumpMem

Outputs the contents of a range of memory cells to the terminal, formatted in hexadecimal.

Receives:

ESI = starting address (first memory cell to be output)

EBX = unit size (1,2, or 4 bytes)

ECX = number of units to output

Returns:

None

Example Code:

```
.data
 myArray DWORD 1,2,3,4,5,6,7,8,9,0Ah,0Bh

.code
 MOV  ESI, OFFSET myArray    ; starting OFFSET (first element of myArray)
 MOV  EBX, TYPE myArray      ; TYPE DWORD = 4 bytes
 MOV  ECX, LENGTHOF myArray  ; Full length of myArray (output all elements)
 CALL DumpMem
```

DumpRegs

Displays contents of the EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP, and EFL (EFLAGS) registers to the terminal, formatted in hexadecimal.

Receives:

None

Returns:

None

Example Code:

```
CALL DumpRegs      ; What have you got there...?
```

GetCommandTail

Copies the program's command tail into a null-terminated string. The "command tail" is the string of characters follow the command/program in the terminal.

Receives:

EDX = address of ASCII buffer to write to, must be at least 129 bytes

Returns:

None

Example Code:

```
.data
 cmdTail BYTE 129 DUP(0)      ; empty buffer, 128 characters + NULL

.code
 MOV  EDX, OFFSET cmdTail
 CALL GetCommandTail          ; With what options did I run this program?
```

GetDateTime

Gets current local date and time, stored in a 64-bit integer in Win32 FILETIME format.

Receives:

PTR QWORD - reference to store datetime

Returns:

PTR QWORD - date and time in FILETIME format

Example Code:

```
.data
 time         QWORD ?
 tenMil       DWORD 10000000
 shift        REAL8 11644473600.
 unixTime     DWORD ?

.code
 PUSH  OFFSET time
 CALL  GetDateTime

; convert Win32 FILETIME to Unix time
 FINIT
 FILD  time
 FIDIV tenMil
 FSUB  shift
 FIST  unixTime

; display time
 MOV   EAX, unixTime ; Decode with unixtimestamp.com
 CALL  WriteDec
```

GetMaxXY

Gets the size of the terminal window's buffer.

Receives:

None

Returns:

AX = number of rows ; max 65,535

DX = number of columns ; max 65,535

Example Code:

```
.data
 numRows     WORD    ?
 numCols     WORD    ?

.code
 CALL GetMaxXY
 MOV  numRows, AX
 MOV  numCols, DX
```

GetMseconds

Gets the number of milliseconds elapsed since midnight on the host computer.

Receives:

None

Returns:

EAX = time since midnight, in milliseconds

Example Code:

```
.data
  startTime    DWORD    ?

.code
  CALL GetMseconds       ; Get Starting Time
  MOV  startTime, EAX    ; Starting Time into EAX
  MOV  EAX, 1000         ;
  CALL Delay             ; Wait 1000ms
  CALL GetMseconds       ; Get Ending Time
  SUB  EAX, startTime    ; EAX should equal approximately 1000... does it?
```

GetTextColor

Gets the current foreground and background colors of the terminal.

Receives:

None

Returns:

AL = terminal color; upper 4 bits = background, lower 4 bits = foreground (text)

Example Code:

```
.data
 color   BYTE   ?

.code
 CALL GetTextColor
 MOV  color, AL
```

GotoXY

Move the cursor at a given row and column in the terminal.

Receives:

DH = X coordinate (column)

DL = Y coordinate (row)

Returns:

None

Example Code:

```
MOV  DH,10          ; row 10
MOV  DL,20          ; column 20
CALL GotoXY         ; move cursor to position (10,20)
```

IsDigit

Determines whether the value in AL is the ASCII code for a valid decimal digit.

Receives:

AL = some value (will be interpreted as ASCII)

Returns:

Zero flag is set if AL is a numerical digit, else Zero flag is clear

Example Code:

```
MOV  AL, someChar
CALL IsDigit
```

MsgBox

Displays a popup message box with an optional title.

Receives:

EDX = address of message string

EBX = address of title string; 0 for blank

Returns:

None

Example Code:

```
.data
msgTitle BYTE "Dialog Title",0
msgHello BYTE "This is a pop-up message box.",10,13
         BYTE "Click OK to continue...",0

.code
 MOV  EBX, OFFSET msgTitle
 MOV  EDX, OFFSET msgHello
 CALL MsgBox
```

MsgBoxAsk

Displays a popup message box with an optional title and Yes/No buttons.

Receives:

EDX = address of question string

EBX = address of title string; 0 for blank

Returns:

EAX = user-selected response; IDYES (6) or IDNO (7)

Example Code:

```
.data
msgTitle    BYTE "Survey Completed",0
msgQuestion BYTE "Thank you for completing the survey.",10,13
            BYTE "Would you like to receive the results?",0

.code
 MOV  EBX, OFFSET msgTitle
 MOV  EDX, OFFSET msgQuestion
 CALL MsgBoxAsk
 CMP  EAX, IDYES  ; check return value in EAX
 JE   _serveYes
 ; User entered 'No'
 JMP _done
_serveYes:
 ; User entered 'Yes'
_done:
```

OpenInputFile

Opens an existing file for input.

Receives:

EDX = address of filename

Returns:

EAX = file handle; INVALID_HANDLE_VALUE if failed to open file

Example Code:

```
.data
fileName  BYTE  "myfile.txt",0

.code
 MOV  EDX, OFFSET fileName
 CALL OpenInputFile
```

ParseDecimal32

Interprets an ASCII string as an unsigned decimal number and converts it to 32-bit binary.

Receives:

ECX = length of string

EDX = address of string

Returns:

EAX = parsed integer

Example Code:

```
.data
buffer  BYTE   "8193"
bufSize = ($ - buffer)

.code
 MOV  EDX, OFFSET buffer
 MOV  ECX, bufSize
 CALL ParseDecimal32 ; Converted value goes into EAX
```

ParseInteger32

Interprets an ASCII string as a signed decimal integer and converts it to 32-bit binary.

Receives:

ECX = length of string

EDX = address of string

Returns:

EAX = parsed integer

Example Code:

```
.data
buffer  BYTE    "-8193"
bufSize = ($ - buffer)

.code
 MOV  EDX, OFFSET buffer
 MOV  ECX, bufSize
 CALL ParseInteger32 ; Converted value goes into EAX
```

Random32

Generates and returns a 32-bit random integer.

Receives:

None

Returns:

EAX = Random Integer

Example Code:

```
.data
randVal DWORD ?

.code
 CALL Randomize
 CALL Random32
 MOV  randVal, EAX ; Move generated value into EAX
```

Randomize

Initializes the starting seed value of the Random32 and RandomRange procedures.

Receives:

None

Returns:

None

Example Code:

```
.data
randVal DWORD ?

.code
 CALL Randomize
 CALL Random32
 MOV  randVal, EAX ; Move generated value into EAX
```

RandomRange

Produces a random integer within a range.

Receives:

EAX = Upper Limit (Exclusive, e.g. this value is not included in the possible values)

Returns:

EAX = Random Integer

Example Code:

```
.data
randVal DWORD ?

.code
 CALL Randomize
 MOV  EAX, 5000
 CALL RandomRange     ; Value will be in the range [0 - 4999]
 MOV  randVal, EAX
```

ReadChar

Reads a single character from the terminal (keyboard-entry).

Receives:

None

Returns:

AL = ASCII Character

AH = Scan Code (optional, if extended key pressed)

Example Code:

```
.data
myChar  BYTE   ?

.code
 CALL ReadChar
 MOV  myChar, AL
```

ReadDec

Reads a 32-bit *unsigned* decimal value from the terminal (keyboard-entry).

Receives:

None

Returns:

EAX = User-entered *unsigned* decimal value

CF = 1 if value is zero or invalid; otherwise **CF** = 0

Example Code:

```
.data
decVal   DWORD   ?

.code
 CALL ReadDec
 MOV  decVal, EAX
```

ReadFloat

Read a *floating-point* value from the terminal (keyboard-entry), push it onto the FPU stack.

Receives:

None

Returns:

ST(0) = User-entered *floating-point* value

Example Code:

```
FINIT                ; initialize FPU
CALL ReadFloat       ; get first float
CALL ReadFloat       ; get second float
FMUL ST(0), ST(1)    ; multiply first and second floats
CALL WriteFloat      ; Write out result
```

ReadFromFile

Reads an input disk file into a memory buffer.

Receives:

EAX = file handle (must already be opened – see OpenInputFile)

ECX = buffer size

EDX = address of buffer (where data is read to)

Returns:

EAX = User-entered *floating-point* value

CF = Error indicator (set if error occurred)

Example Code:

```
.data
BUFFER_SIZE = 5000
fileBuffer  BYTE   BUFFER_SIZE DUP(?)
bytesRead   DWORD  ?

.code
 MOV  EAX, fileHandle          ; already-open file handle
 MOV  EDX, OFFSET fileBuffer   ; points to buffer
 MOV  ECX, BUFFER_SIZE         ; max bytes to read
 call ReadFromFile             ; read the file
```

ReadHex

Reads a 32-bit hexadecimal integer from the terminal (keyboard-entry).

Receives:

None

Returns:

EAX = hexadecimal integer

Example Code:

```
.data
hexVal   DWORD   ?

.code
 CALL ReadHex
 MOV  hexVal, EAX
```

ReadInt

Reads a 32-bit *signed* integer from the terminal (keyboard-entry).

Receives:

None

Returns:

EAX = 32-bit *signed* integer

Example Code:

```
.data
intVal   DWORD   ?

.code
 CALL ReadInt
 MOV  intVal, EAX
```

ReadKey

Performs a no-wait keyboard check to see if any key has been pressed.

Receives:

None

Returns:

AL = ASCII character code, or 0 (if special key)

AH = scan code (if AL = 0)

DX = virtual key code (if AL = 0)

EBX = keyboard flag bits (if AL = 0)

ZF = 0 (key pressed) or 1 (no key)

Example Code:

```
.data
pressedKey   BYTE   ?

.code
 CALL ReadKey
 CMP  ZF, 0            ; Was key pressed?
 JNE  NoKey            ; Key not pressed
 MOV  pressedKey, AL   ; Store entered character
 ; Don't forget to jump over _NoKey
_NoKey:
 ; Do Stuff
```

ReadString

Reads a string from the terminal (keyboard entry), stopping when the Enter key is pressed.

Receives:

ECX = buffer size (max length – 1), to leave space for NULL terminator

EDX = address of buffer (destination of string)

Returns:

EAX = number of bytes (characters) read from terminal

EDX = Address of user-entered string

Example Code:

```
.data
myString    BYTE   21 DUP(0) ; input buffer
byteCount   DWORD ?          ; holds counter

.code
 MOV   EDX, OFFSET myString      ; point to the buffer
 MOV   ECX, SIZEOF myString - 1  ; specify max characters
 CALL ReadString                 ; read string from keyboard
 MOV   byteCount, EAX            ; EAX gets number of characters
```

SetTextColor

Sets the foreground and background colors for text output.

Receives:

EAX = colors (See example code for usage). Color Values are…

black	0
blue	1
green	2
cyan	3
red	4
magenta	5

brown	6
lightGray	7
gray	8
lightBlue	9
lightGreen	10
lightCyan	11

lightRed	12
lightMagenta	13
yellow	14
white	15

Example Code:

```
; SOURCE OPERAND FORMAT: [foreground color] + [background color] * 16)

 MOV EAX, white + (blue * 16)    ; white text on blue background
 CALL SetTextColor
```

ShowFPUStack

Display the contents of the FPU stack.

Receives:

None

Returns:

None

Example Code:

```
.data
oneFloat    REAL8 123.456
twoFloat    REAL8 10.0

.code
 FINIT                  ; initialize FPU
 FLD    oneFloat        ; load first value
 FLD    twoFloat        ; load second value
 CALL   ShowFPUStack    ; show FPU stack contents
```

Str_Compare

Compares two strings, setting the Zero and Carry flags. Parameters are passed on stack; see Example Code for usage.

Receives:

Stack Param1 = Offset to First String

Stack Param2 = Offset to Second String

Returns:

ZF=1 CF=0 if string1 = string2

ZF=0 CF=0 if string1 > string2

ZF=0 CF=1 if string1 < string2

Example Code:

```
.data
stringA   BYTE "abcde", 0
stringB   BYTE "xyz", 0

.code
 PUSH OFFSET stringA
 PUSH OFFSET stringB
 CALL Str_Compare
 JE   _sameString      ;stringA = stringB
 JA   _largerSA        ;stringA > stringB
 JB   _smallerSA       ;stringA < stringB
```

Str_Copy

Copy a string. Parameters are passed on stack; see Example Code for usage.

Receives:

Stack Param1 = Offset to Source String

Stack Param2 = Offset to Target String

Returns:

Stack Param2 = Offset to Target String (now equal to Source String)

Example Code:

```
.data
oldString   BYTE "abcde",0
newString   BYTE LENGTHOF oldString dup(0)

.code
 PUSH OFFSET newString
 PUSH OFFSET oldString
 CALL Str_copy            ; newString = oldString
```

Str_Length

Returns the length of a null-terminated string.

Receives:

EDX = Address Offset to String

Returns:

EAX = Length of Reference String

Example Code:

```
.data
myString    BYTE  "abcde",0
strLength   DWORD ?

.code
 MOV  EDX, OFFSET myString  ; point to string
 CALL Str_Length            ; EAX = 5 (null term not counted)
 MOV  strLength, EAX        ; save length
```

Str_Trim

Removes occurrences of a character from the end of a string. Parameters are passed on stack; see Example Code for usage.

Receives:

Stack Param1 = Character to remove

Stack Param2 = Offset to Target String

Returns:

Stack Param2 = Offset to Target String (trimmed)

Example Code:

```
.data
myString    BYTE "abcde###",0
myChar      BYTE '#'

.code
 PUSH myChar
 PUSH OFFSET myString
 CALL Str_Trim
```

Str_UCase

Converts entire string to upper case. Parameters are passed on stack; see Example Code for usage.

Receives:

Stack Param1 = Offset to Target String to be Converted to Upper Case

Returns:

Stack Param1 = Offset to Target String (Converted)

Example Code:

```
.data
myString  BYTE  "abcde", 0

.code
 PUSH OFFSET myString
 CALL Str_UCase          ; now "ABCDE",0
```

WaitMsg

Displays the message “Press any key to continue. . .” and waits for the user to press a key.

Receives:

None

Returns:

None

Example Code:

```
CALL  WaitMsg   ; yeah... pretty simple
```

WriteBin

Writes a 32-bit integer to the terminal in ASCII binary format.

Receives:

EAX = Integer to be written

Returns:

None

Example Code:

```
MOV  EAX, 1234ABCDh
CALL WriteBin
; Output: 0001 0010 0011 0100 1010 1011 1100 1101
```

WriteBinB

Writes an unsigned 8, 16, or 32-bit number to the terminal in binary format.
EBX must contain the TYPE of the number to write (1 for BYTE, 2 for WORD, or 4 for DWORD)

Receives:

AL, AX, or EAX = Integer to be written
EBX = 1 (write AL as 8-bit binary) OR
EBX = 2 (write AL as 16-bit binary) OR
EBX = 4 (write AL as 32-bit binary)

Returns:

None

Example Code:

```
MOV  EAX, 0Ah    ; one byte of data
MOV  EBX, 1      ; Write as one byte
CALL WriteBinB
; Output: 0000 1010
```

WriteChar

Write a single ASCII character to the terminal.

Receives:

AL = ASCII character to be written

Returns:

None

Example Code:

```
MOV  AL, '-'
CALL WriteChar
; Output: -
```

WriteDec

Displays a 32-bit *unsigned* integer to output.

Receives:

EAX = Integer to be written

Returns:

None

Example Code:

```
MOV  EAX, 186
CALL WriteDec
; Output: 186
```

WriteFloat

Write the floating-point value from ST(0) to the output.

Receives:

ST(0) = Float to be written (Top of the FP Stack)

Returns:

None

Example Code:

```
.data
myFloat  REAL8  25.0

.code
 FINIT             ; initialize FPU
 FLD  myFloat      ; push myFloat to ST(0)
 CALL WriteFloat
```

WriteHex

Writes the contents of EAX as an 8-digit hexadecimal value.

Receives:

EAX = Value to be written

Returns:

None

Example Code:

```
MOV  EAX, 7FFFh
CALL WriteHex
; Output: 00007FFF
```

WriteHexB

Writes an unsigned 8, 16, or 32-bit number to the terminal in hexadecimal format.
EBX must contain the TYPE of the number to write (1 for BYTE, 2 for WORD, or 4 for DWORD)

Receives:

AL, AX, or EAX = Integer to be written

EBX = 1 (write AL as 2 hex digits) OR

EBX = 2 (write AL as 4 hex digits) OR

EBX = 4 (write AL as 8 hex digits)

Returns:

None

Example Code:

```
MOV  EAX, 0A0Bh    ; two bytes of data
MOV  EBX, 2        ; Write as two bytes
CALL WriteHexB
; Output: 0A0B
```

WriteInt

Displays a 32-bit *signed* integer to output. Prepends a sign character (+ or -)

Receives:

EAX = Integer to be written

Returns:

None

Example Code:

```
MOV  EAX, 186
CALL WriteInt  ; Output: +186
CALL CrLf
MOV  EAX, -186
CALL WriteInt  ; Output: -186
```

WriteString

Writes a null-terminated string to the terminal.

Receives:

EDX = Address offset of string to be written.

Returns:

None

Example Code:

```
.data
myString  BYTE  "Enter your favourite colour: ", 0

.code
 MOV  EDX, OFFSET myString
 CALL WriteString        ; Output: Enter your favourite colour:
 CALL Crlf
```

WriteToFile

Writes the contents of a buffer to an output file.

Receives:

EAX = Operating System File Handle

ECX = Number of bytes to write.

EDX = Address offset of buffer to be written.

Returns:

If **CF = 0: EAX** = Number of bytes written.

If **CF = 1: EAX** = Error code. Call WriteWindowsMsg to see error.

Example Code:

```
.data
BUFSIZE = 5000
buffer        BYTE  BUFSIZE DUP(0)
bytesWritten  DWORD 0

.code
 MOV  EAX, fileHandle
 MOV  EDX, OFFSET buffer
 MOV  ECX, BUFSIZE
 CALL WriteToFile
 JC   _showError
 MOV  bytesWritten, EAX
```

WriteWindowsMsg

Writes a string containing the most recent error generated by your application to the output when executing a call to a system function.

Receives:

None

Returns:

None

Example Code:

```
CALL WriteWindowsMsg
```