# Lab 13:

# Reading and writing files:

## CreateFile Function:

The CreateFile function either creates a new file or opens an existing file. If successful, it returns a handle to the open file; otherwise, it returns a special Constant name INVALID\_HANDLE\_VALUE.

Here is the prototype:

**Call args:** EDX contains the offset of a filename

**Return arg:** EAX contains a valid file handle if the operation was successful.

Otherwise, EAX contains INVALID\_HANDLE\_VALUE.

**Example:**

mov edx,OFFSET filename

call CreateOutputFile

mov filehandle, EAX

## ReadFile Function

The ReadFile function reads text from an input file. Here is the prototype:

**Call args:** EAX = an open file handle

EDX = offset of the input buffer

ECX = maximum number of bytes to read

**Return arg:**

If CF = 0, EAX contains the number of bytes read.

If CF = 1, EAX contains a system error code. Call WriteWindowsMsg

to get a text representation of the error.

**Example:**

.data

BUFSIZE = 5000

buffer BYTE BUFSIZE DUP(?)

bytesRead DWORD ?

.code

mov eax,fileHandle

mov edx,OFFSET buffer

mov ecx,BUFSIZE

call ReadFromFile

jc show\_error\_message

mov bytesRead,eax

## WriteFile Function

The WriteFile function writes data to a file, using an output handle. The handle can be the

screen buffer handle, or it can be one assigned to a text file. The function starts writing data to

the file at the position indicated by the file’s internal position pointer. After the write operation has

been completed, the file’s position pointer is adjusted by the number of bytes actually written. Here

is the function prototype:

**Call args:** EAX = an open file handle

EDX = offset of the buffer

ECX = maximum number of bytes to write

**Return arg:**

If CF = 0, EAX contains the number of bytes written.

If CF = 1, EAX contains a system error code. Call WriteWindowsMsg

to get a text representation of the error.

**Example:**

.data

BUFSIZE = 5000

buffer BYTE BUFSIZE DUP(?)

bytesWritten DWORD ?

.code

mov eax,fileHandle

mov edx,OFFSET buffer

mov ecx,BUFSIZE

call WriteToFile

jc show\_error\_message

mov bytesWritten,eax

## CloseFile Function

**Call args:** EAX contains an open file handle

**Return arg:** EAX is nonzero if the operation was successful

**Example:**

mov eax,fileHandle

call CloseFile

# MWrite

Writes a string literal (no terminal null) to standard output.

**Parameter:**  
  
   **text:REQ** - A string literal.

**Example:**

mWrite **"Hello World"**

**Dependencies:**  
  
   Causes a call to the [WriteString](https://csc.csudh.edu/mmccullough/asm/help/source/irvinelib/writestring.htm) procedure.

# DumpMem

Writes a range of memory to standard output in hexadecimal.

**Call args:** ESI = starting offset

ECX = number of units

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

**Return arg:** None

**Example:** Dump a word array

.data

array WORD 8,9,10,11,0FFFFh

.code

mov esi,OFFSET array

mov ecx,LENGTHOF array ;5

mov ebx,TYPE array ;2

call DumpMem

# Stack Applications

There are several important uses of runtime stacks in programs:

• A stack makes a convenient temporary save area for registers when they are used for more

than one purpose. After they are modified, they can be restored to their original values.

• When the CALL instruction executes, the CPU saves the current subroutine’s return address

on the stack.

• When calling a subroutine, you pass input values called arguments by pushing them on the stack.

• The stack provides temporary storage for local variables inside subroutines

## PUSH Instruction

The PUSH instruction first decrements ESP and then copies a source operand into the stack.

A 16-bit operand causes ESP to be decremented by 2. A 32-bit operand causes ESP to be decremented by 4. There are three instruction formats:

PUSH reg/mem16

PUSH reg/mem32

PUSH imm32

## POP Instruction

The POP instruction first copies the contents of the stack element pointed to by ESP into a 16- or

32-bit destination operand and then increments ESP. If the operand is 16 bits, ESP is incremented by 2; if the operand is 32 bits, ESP is incremented by 4:

POP reg/mem16

POP reg/mem32

**Tasks:**

1. Write an assembly program that takes input of filename to be created and creates file of that name. After that write the following text in file using your program:

*Hello my name is {your name}. My hometown is {your hometown}. My roll number is {your rollnumber}. I love assembly language and I don’t mind repeating it.*

1. Write an assembly program opens the above created filed and does the following:
   1. Read the file
   2. Calculate number of vowels and consonants in the above file content.
   3. Count number of words in the above file
   4. Extract your name, roll number and hometown from filecontent (you may use hardcoded indexes to do that).
   5. Display them on console.
2. Write an assembly program that calculates factorial of a given number using stacks and recursion.

**Submission Instructions:**

* Write your name, roll no and section on top of your code file
* Do all of your code work in procedures and just call that procedure in main to execute the code.

Eg:

Q1 proc

;code

ret

Q1 endP

* **Self-Evaluation: (bonus 2 marks)**

You are to self-evaluate all the questions you do. Write a one-liner comment for each question on top of code file below your name.

Eg. Q1, all done, works completely fine for all use cases

Q2, partially done, exception thrown error

**You must be honest in self-evaluating yourself.**

**If during evaluation you are found being dishonest, your lab will be marked zero.**

* Submit only one .asm file (Format: i22-1234\_LAB13.asm