**Practical 13**

**Assembly Language and DLLs**

Objectives:

* Understand Little Endian byte order
* Recognize the difference a program written in assembly and in C
* Describe the steps to display information in assembly
* Use the debug command to trace an assembly program
* Write a simple addition, subtraction and multiplication program in assembly
* Explain the security issues associated with DLLs

**Exercise Using Hex Editors**

In Win10 VM:

1. Using Cygwin, modify your HelloWorld.c program from the previous Practical on Decompilers Exercise 1 so that it looks like the following:

#include <stdio.h>

int main () {

printf("Hello World\n");

printf("This is a good day!!\n");

}

1. Compile it. The executable “HelloWorld.exe” will be generated.

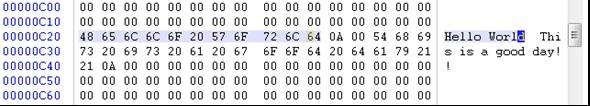
gcc –o HelloWorld HelloWorld.c

1. Run the program

./HelloWorld.exe

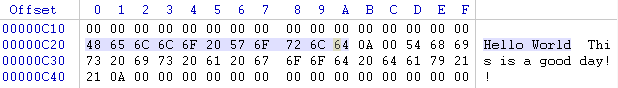
1. Download and install a hex editor (eg WinHex or HxD Hex Editor). Both hex editors are also available on Brightspace or the usual download link, under Files-for-Topic13 :
2. Use the hex editor to open HelloWorld.exe (the file would be located in C:\cygwin64\home\<username>)
3. The contents of the executable program is displayed : both in hexadecimal format and Ascii format.
4. Scroll down the program. Are you able to see the strings “Hello World” and “This is a good day!!” in the ASCII portion (right hand side)? Using your mouse, select the string “Hello World”. The corresponding hexadecimal characters that represent “Hello World” will be selected too.

These characters are “Hello World” in hexadecimal format



1. The offset column indicates the number of bytes offset from the start of the program. For example, in the example below, the byte “48h” (the character “H”) has the offset of C20h from the start of the program, and the next byte “65h” (the character “e”) has the offset of C21h.

You will have different offset values from this example



1. What is the offset of the byte “72h”?

What is the offset of the byte “68h”?

What is the offset of the byte “67h”?

In the example above, the offset of byte “72h” is C28 (in hexadecimal)

In the example above, the offset of byte “68h” is C2E (in hexadecimal)

In the example above, the offset of byte “67h” is C37 (in hexadecimal)

**Exercise Little Endian byte order**

Numbers are stored using Little Endian byte order (smaller bytes in lower addresses) on systems with Intel PC processors.

In Kali (or Win10 VM):

1. Create the following C program “data.c”.

void main() {

int myinteger = 2864434397;

char mystring[10] = "ABCDEFGH";

}

1. What is the hex equivalent of 2864434397? (You can use the calculator to get the answer)

Ans : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Ans : AABBCCDD)

1. Compile the C program. You may get a warning about unsigned int – you can ignore it.

gcc –o data data.c

1. If you are using Kali, look at the manual page for the in-built hex editor.

man hexeditor

You can use the PageUp, PageDown or arrow keys to scroll through a file. To do a Search, press Control-W.

1. Use the hex editor to open the compiled program.

hexeditor data

1. To do a search for data bytes, in Kali hexeditor, press Control-W. In WinHex or HxD, go to Search, Find, Hex Values. Search forward for the hexadecimal string “414243”. This is the first three hex characters of the variable mystring “ABCD...”.
2. Look for the bytes containing “DD CC BB AA”. This is the variable myinteger – with its contents stored in Little Endian (least significant bytes stored in lowest addresses).



Complete the memory address table below for the 4 bytes of myinteger:

|  |  |
| --- | --- |
| Address offset | Data |
| 41D | DD |
| 41E | CC |
| 41F | BB |
| 420 | AA |

Following the above example, the memory address table for the 4 bytes of myinteger would be as follows. You may have different values for your system.

|  |  |
| --- | --- |
| Address offset | Data |
| 41D | DD |
| 41E | CC |
| 41F | BB |
| 420 | AA |

1. Press Control-C to quit the Kali hex editor.

**Exercise Assembly vs C**

In Win10 VM:

1. Modify the HelloWorld.c so that it prints only “Hello World!”.

#include <stdio.h>

int main () {

printf("Hello World!");

}

1. Compile it. Take note of the file size of the generated HelloWorld.exe (it should be located in C:\cygwin64\home\<username>).
2. Run the HelloWorld.exe to check its output.

In this practical, we will create some Assembly programs to get a basic understanding of them. We will start by creating simple 16-bit assembly programs on the old WinXP virtual machine.

In WinXP VM:

1. Create a folder C:\asm for storing your assembly programs.
2. Download the Turbo Assembler (tasm.exe) and Turbo Linker (tlink.exe) from Brightspace or the download link to C:\asm.

To copy and paste files to the WinXP VM, you can go to the VM menu, and Install VMware Tools. Install the VMware Tools using default options. Restart the WinXP VM.

1. Create the following text file C:\asm\hello.asm.

; my first assembly program

dosseg

.model small

.stack 100h

.data

msg db 48h,65h,6ch,108,6fh,2ch,20h,57h,6fh,72h,6ch,64h,'!','$'

.code

main proc

mov ax,@data

mov ds,ax

mov ah,9

mov dx,offset msg

int 21h

mov ax,4C00h

int 21h

main endp

end main

Note : Assembly instructions following Intel syntax (like in this program) will have the MOV instructions following this format :

MOV dest, source

1. Start a Command Prompt and change directory to the asm folder.

cd c:\asm

1. Compile the assembly program using Turbo Assembler. Note : Turbo Assembler can only accept filenames of max 8 characters and file extensions of max 3 characters.

tasm hello.asm

1. If there are no errors reported, check that a hello.obj file has been created in the folder.
2. Link the object file with Turbo Linker.

tlink hello.obj

1. If there are no errors reported, check that a hello.exe file has been created in the folder.
2. Run the executable.

hello

1. Take note of the file sizes of hello.asm and hello.exe.

**Questions**

Is there any difference between the outputs of the C HelloWorld.exe and the Assembly hello.exe? (Ans : No difference, they both print the words “Hello World!”)

Which has the smaller file size : the C HelloWorld.exe or the Assembly hello.exe?

(The C HelloWorld.exe is about 155,000 bytes in size, while the Assembly hello.exe is about 550 bytes in size)

Which do you think will execute faster : the C HelloWorld.exe or the Assembly hello.exe?

(Ans : Assembly programs will execute faster)

What do you think “48h,65h,6ch,108,6fh,2ch,20h,57h,6fh,72h,6ch,64h,'!','$'” in the code represent? Hint : check the Ascii table

(Ans : The bytes represent the string “Hello, World!”

What do “ax”, “ds” and “dx” represent?

They represent different registers. You can find information on Registers at https://www.csc.depauw.edu/~bhoward/asmtut/asmtut2.html

The code “int 21h” is seen twice in the code. What does each “int 21h” do?

Hint: Browse to http://spike.scu.edu.au/~barry/interrupts.html

(Ans :

For the first int 21h, the AH part of the register AX has the value “09”, so the program will print a string to STDOUT. The address of the string is found in DX register, which is the address of the “Hello, World” string.

For the second int 21h, the AH part of the AX register has the value “4C”, so the program will exit.)

**Exercise Debugging the Assembly code**

In WinXP VM

1. Create another assembly program in c:\asm. (give any filename to the assembly program, eg “try.asm”)

dosseg

.model small

.stack 100h

.code

main proc

mov ax,0020

mov bx,0030

sar ax,1

or ax,bx

nop

main endp

end main

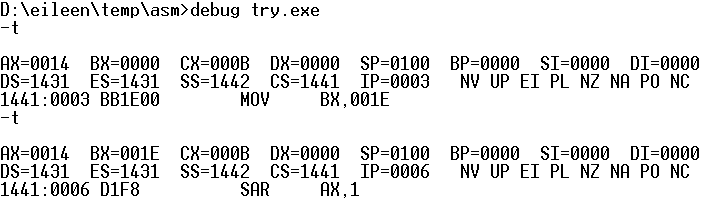
1. Use tasm and tlink to compile and link the code.
2. Use the debugger to load the code.

debug <executable filename>

1. At the dash prompt (-), type ? to view a list of commands.
2. Run a trace to step through line by line of the code.

Each time you enter “t” to trace through the code, the values of the CPU registers will be displayed, along with the next instruction to be run. Observe the values of the AX and BX registers.

The next instruction to be executed



Observe how the values in the AX and BX registers change

1. When you reach the NOP instruction, you can quit the debugger.

**Questions**

What do the assembly instructions “mov”, “sar”, “or” and “nop” do?

<http://www.jegerlehner.ch/intel/IntelCodeTable.pdf>

mov : Copy data from source to destination

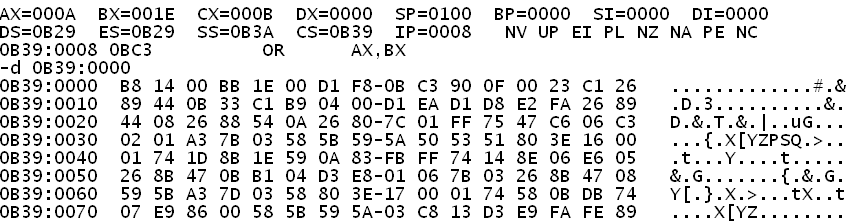
sar : Shift arithmetic right and store the result in AX

or : perform logical OR and store the result in AX

How would I view memory addresses on a computer using the debug tool (you are dumping out memory contents)?

Ans : At the debug prompt, you can type “d” to dump out memory contents.

In the following example, to see the contents of a particular part of memory, the compiled code, you can type “d” followed by the memory address in the Code Segment register (CS)



Can you use the debug on an application compiled by a C compiler?

Ans : You can debug any executable programs written in any programming language.

**Exercise Writing assembly program to add two single digit numbers**

In WinXP VM:

1. Create a new assembly program c:\asm\add.asm.
2. Type in the following assembly code. You need to fill in the blanks!

; assembly program to add two single digit numbers

dosseg

.model small

.stack 100h

.code

main proc

mov ax,4

mov bx,3

add ax,bx ; add the contents of ax and bx

; result will be stored in ax

add ax,30h ; convert number to its ascii code

mov ah,9 ; interrupt code to write a character

; to standard output

mov dl ,al ; copy the character to be displayed

; from lower byte of ax to data register

int 21h

mov ax,4C00h

int 21h

main endp

end main

1. Compile, link and run the program to test it.

**Exercise Use a hex editor to view and modify the executable code**

In WinXP VM:

1. Use a hex editor to open the program “add.exe”. (you can install HxD or WinHex)
2. Look for the bytes with the values 04 and 03. These bytes match the constant values “mov ax,4” and “mov bx,3” in the program.



1. Change the byte with the value 04 to 05. Save add.exe and run it. The program now displays a different result.
2. Using the hex editor, change the byte with the modified value of 05 to 08. Save add.exe and run it. Do you get the correct result?

You should see a semi-colon being displayed – a wrong result.

A picture containing background pattern

Description automatically generated

**Question**

Why did the program display a semi-colon when adding the numbers 8 and 3?

Ans : View the video for Intro to Assembly : Using Hex Editors to modify executable files for the explanation (the link to the videos is in Brightspace)

**Exercise Writing assembly program that adds two single digit numbers and handles two-digit output**

In WinXP VM:

1. Modify add.asm so that it can handle two-digit outputs. (Changes are highlighted in blue)

; assembly program to add two single digit numbers

dosseg

.model small

.stack 100h

.code

main proc

mov ax,**6**

mov bx,**7**

add ax,bx ; add the contents of ax and bx

; result will be stored in ax

**mov bx,ax ; result copied to bx**

**sub bx,9 ; subtract 9 from bx and store result in bx**

**jg longz ; if previous instruction result is**

**; more than zero, then result is**

**; a two-digit number, jump to longz**

**; else result is one-digit number**

add ax,30h ; convert number to its ascii code

mov ah,\_\_9\_\_ ; interrupt code to write a character

; to standard output

mov \_dl\_\_,al ; copy the character to be displayed

; from lower byte of ax to data register

int 21h

**jmp donez ; one-digit result displayed, jump to end**

**longz:**

**mov bx,ax ; copy result to bx**

**mov dl,31h**

**mov ah,\_\_9\_\_ ; interrupt code to write a character**

**; to standard output**

**int 21h**

**mov ax,bx**

**sub ax,10**

**add ax,30h**

**mov dl,al**

**mov ah,\_\_9\_\_ ; interrupt code to write a character**

**; to standard output**

**int 21h**

**donez:**

mov ax,4C00h

int 21h

main endp

end main

Optional Task

Create a new assembly program c:\asm\sub.asm that subtracts two single digit numbers.

**Exercise Diassemble a program**

In WinXP:

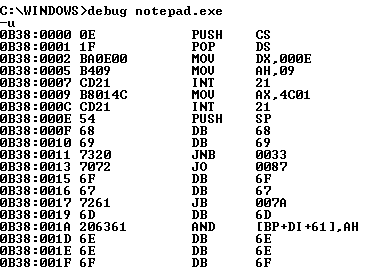
1. In a command prompt, browse to the C:\Windows folder.

cd c:\windows

1. Use Debug to open the Notepad executable.

debug notepad.exe

1. Type “?” to view the list of commands. Note that “U” means unassemble (or disassemble).
2. Type “u” to view a segment of disassembled code. The output may look something like the following :



Disassembled code for the Notepad application (Only the disassembled code in the address range 0B38:0000 to 0B38:001F is listed)

1. If you want to view more disassembled code, specify the address range, for example :

u 0B38:0020

Think about :

Using a debugger program, hex editor and with basic knowledge of assembly, if a user could trace an application and find out where the variables are stored for that application, would the user be able to override memory locations to

1. break the application i.e. crash the program?
2. gather sensitive data i.e. sql statements, database connection strings?
3. overcome access control i.e. login without proper credentials?

Ans : All the above might be possible

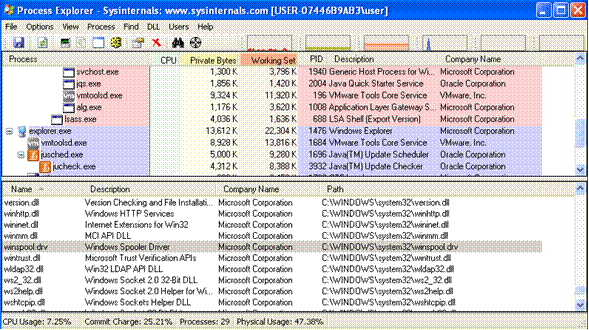
**Exercise List out the DLLs used by a process (STOP HERE)**

In WinXP:

1. Do a search on the Internet for Process Explorer, and download it (A copy is also available Brightspace and in the download link)
2. Unzip the file and run Process Explorer
3. A list of currently running processes is shown. Start a program (eg Internet Explorer or Firefox) and select it in Process Explorer.

List of processes currently running

List of DLLs called by the selected process



1. If you do not see the lower pane, go to View menu, Show Lower Pane.
2. To see the DLLs being called by the process, go to View menu, Lower Pane View, DLLs. (Handles will list out the objects that are currently opened by the process, eg files)

**Exercise DLL Search Order**

1. When an application like Internet Explorer needs to load a DLL, where does it look for the DLL file?

(Hint : Do an Internet search for “DLL Search Order”. A detailed description can usually be found in msdn.microsoft.com)

1. Based on the search order for a DLL, can someone cause his own DLL to be called by the application?

If the hacker has access to the directories where DLLs are searched, eg C:\Windows\System32, he can overwrite a legitimate DLL file with his own DLL file which contains malicious code. Applications will then call his malicious DLL file.

Some information on DLL Security :

https://docs.microsoft.com/en-us/windows/desktop/dlls/dynamic-link-library-security

~~~ End of Practical ~~~