

CSC 3210
Computer Organization and Programming
Lab 7
Answer Sheet

Student Name: Vivian Do
Section: CRN 90913; 11:00-12:40
Oct 08 2021

Lab 7(a)

Fix the errors in the provided code.
Build and Attach screenshot showing the code and “build succeeded” message.

The screenshot displays the Visual Studio IDE with the following components:

- Assembly Code (main.asm):**

```
1 ; Lab 7a
2 .386
3 .model flat, stdcall
4 .stack 4096
5 ExitProcess proto, dwExitCode:dword
6
7 .data
8     count BYTE 100
9     wVal WORD 2
10    wVal2 word 4
11
12 .code
13 main proc
14     mov bl, count
15     mov ax, wVal
16     mov count, al
17
18     mov ax, wVal      ; changed al to ax (originally was mov al, wVal)
19     mov al, count     ; changed ax to al (originall was mov ax, count)
20     mov al, count     ; changed eax to al (originall was mov eax, count)
21     mov ax, wVal2     ; changed switched wVal with wVal2 and then moved wVal2 to ax (originally was mov wVal2, wVal)
22     mov ax, wVal      ; added another instruction to mov wVal to ax
23     invoke ExitProcess, 0
24
25 main endp
26 end main
27
```
- Output Window:**

```
Show output from: Build
Build started...
1>----- Build started: Project: Lab7a, Configuration: Debug Win32 -----
1>Assembling main.asm...
1>Lab7a.vcxproj -> C:\Users\vivia\source\repos\Lab7a\Debug\Lab7a.exe
***** Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped *****
```
- Status Bar:** 100 % No issues found Ln: 27 Ch: 1 TABS CRLF
- Bottom Bar:** Build succeeded

Lab 7(b)

Debug through each line of instructions.
Take screenshot that includes code and register window.
Record the register content.

and explain the register contents.

Code:

```
main.asm  -  X
1  ; Lab 7b
2  .386
3  .model flat, stdcall
4  .stack 4096
5  ExitProcess proto dwExitCode:dword
6
7  .data
8      myByte1 BYTE 9Bh
9  .code
10 main proc
11     mov bx, 0A69Bh
12     movzx eax, bx
13                                     ; EAX = 0000A69B
14     movzx eax, myByte1
15                                     ; EAX = 0000009B
16     mov bx, 0A69Bh
17     movsx eax, bx
18                                     ; EAX = FFFFA69B
19     invoke ExitProcess, 0
20
21 main endp
22 end main
23
```

Line number: 11

Instruction: `mov bx, 0A69Bh`

Register Values: `EBX = 0000A69B`

Screenshot:

```
Registers
EAX = 001EFA8C EBX = 0053A69B ECX = 002F1005 EDX = 002F1005 ESI = 002F1005 EDI = 002F1005 EIP = 002F1014 ESP = 001EFA34 EBP = 001EFA40 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
```

Explanation: EBX register is 32-bit long with an unsigned integer variable. This register is updated with 0A69Bh in its contents by `mov`.

Line number: 12

Instruction: `movzx eax, bx`

Register Values: `EAX = 0000A69B`

Screenshot:

```
Registers
EAX = 0000A69B EBX = 005BA69B ECX = 002F1005 EDX = 002F1005 ESI = 002F1005 EDI = 002F1005 EIP = 002F1017 ESP = 006FFCA4 EBP = 006FFCB0 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
|
```

Explanation: EAX register is 32-bit long with a signed integer variable. This register is updated with EBX (0A69Bh) while extending the remaining upper half with zeros.

(Copy paste this format if you need more)

Line number: 14

Instruction: `movzx eax, myByte1`

Register Values: EAX = 0000009B

Screenshot:

```
Registers
EAX = 0000009B EBX = 0053A69B ECX = 002F1005 EDX = 002F1005 ESI = 002F1005 EDI = 002F1005 EIP = 002F101E ESP = 001EFA34 EBP = 001EFA40 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
|
```

Explanation: The EAX register is updated with myByte1 value (9Bh) while extending the remaining upper half with zeros.

Line number: 16

Instruction: `mov bx, 0A69B`

Register Values: EAX = 0000A69B

Screenshot:

```
Registers
EAX = 0000009B EBX = 0053A69B ECX = 002F1005 EDX = 002F1005 ESI = 002F1005 EDI = 002F1005 EIP = 002F1022 ESP = 001EFA34 EBP = 001EFA40 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
|
```

Explanation: Nothing has changed. Refer back to the explanation for line 11.

Line number: 17

Instruction: `movsx eax, bx`

Register Values: EAX = FFFFA69B

Screenshot:

```
Registers
EAX = FFFFA69B EBX = 0053A69B ECX = 002F1005 EDX = 002F1005 ESI = 002F1005 EDI = 002F1005 EIP = 002F1025 ESP = 001EFA34 EBP = 001EFA40 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
|
```

Explanation: The EAX register is updated with the contents from the bx register. The leading A tells us that it is the highest bit in set. The rest of the upper half of the register is extended with F's.

Lab 7(c)

Debug through each line of instructions.
Take screenshot that includes code and register window.
Record the register content.
and explain the register contents.

Code:

```
main.asm  X
1  .386
2  .model flat, stdcall
3  .stack 4096
4  ExitProcess proto dwExitCode:dword
5
6  .data
7      arrayD DWORD 10000h, 20000h, 30000h
8
9  .code
10 main proc
11     ; direct-offset addressing (doubleword array):
12     mov eax, arrayD           ; EAX = 00010000
13     mov ebx, [arrayD + 4]     ; EBX = 00020000
14     mov edx, [arrayD + 8]     ; EDX = 00030000
15     invoke ExitProcess, 0
16
17 main endp
18 end main
19
```

Line number: 12

Instruction: `mov eax, arrayD`

Register Values: `EAX = 00010000`

Screenshot:

```
Registers
EAX = 00010000 EBX = 00BFB000 ECX = 00E11005 EDX = 00E11005 ESI = 00E11005 EDI = 00E11005 EIP = 00E11015 ESP = 00D5FA68 EBP = 00D5FA74 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
0x00E14004 = 00020000 |
```

Explanation: EAX register is 32-bit long with a signed integer variable. This register is updated with arrayD (10000h) in its contents by `mov`.

Line number: 13

Instruction: `mov ebx, [arrayD + 4]`

Register Values: `EBX = 00020000`

Screenshot:

```
Registers
EAX = 00010000 EBX = 00020000 ECX = 00E11005 EDX = 00E11005 ESI = 00E11005 EDI = 00E11005 EIP = 00E11018 ESP = 00D5FA68 EBP = 00D5FA74 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
|
0x00E14008 = 00030000
```

Explanation: EBX register is 32-bit long with an unsigned integer variable. This register is updated with 200000h. By adding 4 to arrayD, we are accessing the array element that is 4 bytes offset to the first one.

Line number: 14

Instruction: `mov edx, [arrayD + 8]`

Register Values: EDX = 00030000

Screenshot:

```
Registers
EAX = 00010000 EBX = 00020000 ECX = 00E11005 EDX = 00030000 ESI = 00E11005 EDI = 00E11005 EIP = 00E11021 ESP = 00D5FA68 EBP = 00D5FA74 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
```

Explanation: EDX register is updated with 300000h. By adding 8 to arrayD, we are accessing the array element that is 8 bytes away from the first byte.

Lab 7(d)

Create a new project to run the following program.

Declare an array in the data segment: `arrayB WORD 1,2,3,4`

Write code to Rearrange the array as follows: 4,3,1,2

Add the screenshot of your code here.

The registers EAX, EBX, ECX, and EDX at the end show the rearranged array.

The screenshot shows a debugger window with the 'Registers' tab at the top. The register values are: EAX = 00AF0004, EBX = 00820003, ECX = 00D60002, EDX = 00D60001, ESI = 00D61005, EDI = 00D61005, EIP = 00D61058, ESP = 00AFFC44, EBP = 00AFFC50, EFL = 00000246. Below the registers, the status bar shows 'OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0'. The main window displays assembly code for 'main.asm'. The code includes a data segment with 'arrayB WORD 1,2,3,4' and a code segment with a 'main' procedure. The procedure contains instructions for swapping elements in 'arrayB' and checking the results. A comment on line 27 says 'invoke ExitProcess, 0' and a status bar at the bottom indicates 'No issues found'.

```
Registers
EAX = 00AF0004 EBX = 00820003 ECX = 00D60002 EDX = 00D60001 ESI = 00D61005 EDI = 00D61005 EIP = 00D61058 ESP = 00AFFC44 EBP = 00AFFC50 EFL = 00000246
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0

100 %
main.asm
1 ; Lab 7d
2 .386
3 .model flat, stdcall
4 .stack 4096
5 ExitProcess proto dwExitCode:dword
6
7 .data
8     arrayB WORD 1,2,3,4
9
10 .code
11 main proc
12     mov ax, arrayB
13
14     xchg ax, [arrayB + 6] ; swapping element in arrayB (1) with element in arrayB + 6 (4)
15     mov arrayB, ax
16     mov cx, [arrayB + 6]
17
18     mov ax, [arrayB + 2] ; swapping element in arrayB + 2 (2) with element in arrayB + 4 (3)
19     xchg ax, [arrayB + 4]
20     mov [arrayB + 2], ax
21
22     mov ax, arrayB ; checking to see if the elements are swapped by putting them back in register
23     mov bx, [arrayB + 2]
24     mov cx, [arrayB + 4]
25     mov dx, [arrayB + 6]
26
27     invoke ExitProcess, 0 ; 1ms elapsed
28
29 main endp
30 end main
31

100 % No issues found Ln: 27 Ch: 1 TABS CRLF
```

Lab 7(e)

Create a new application to run the following program.

The data segment is provided:

.data

Val1 SWORD 23

Val2 SWORD -35

Val3 SDWORD 4

Evaluate the following expression:

$EBX = (-Val1 + val2) + (val3 * 3)$

You can only use Mov, Movsz, Movzx, Add, Sub instructions.

Build and run the program using the debugger

Debug the code until you reach "INVOKE ExitProcess, 0" and attach a screenshot of your code and EBX register content.

Registers

EAX = 00EFFF74 EBX = FFFFFFFD2 ECX = 00681005 EDX = 00681005 ESI = 00681005 EDI = 00681005 EIP = 00681036 ESP = 00EFFF1C EBP = 00EFFF28 EFL = 00000286

100 %

main.asm

```
1 ; lab 7e
2 .386
3 .model flat, stdcall
4 .stack 4096
5 ExitProcess proto, dwExitCode:dword
6
7 .data
8     Val1 SWORD 23
9     Val2 SWORD -35
10    Val3 SWORD 4
11
12 .code
13 main proc
14     ; calculating for EBX = (-Val1 + Val2) + (Val3 * 3)
15     mov bx, Val3
16     ; since 4 * 3 = 12, i'm adding Val3 to bx two times
17     add bx, Val3
18     add bx, Val3
19     add bx, Val2
20
21     movsx ebx, bx
22     sub bx, Val1
23     ; EBX = FFFFFFFD2
24     invoke ExitProcess, 0
25
26 main endp
27 end main
28
```