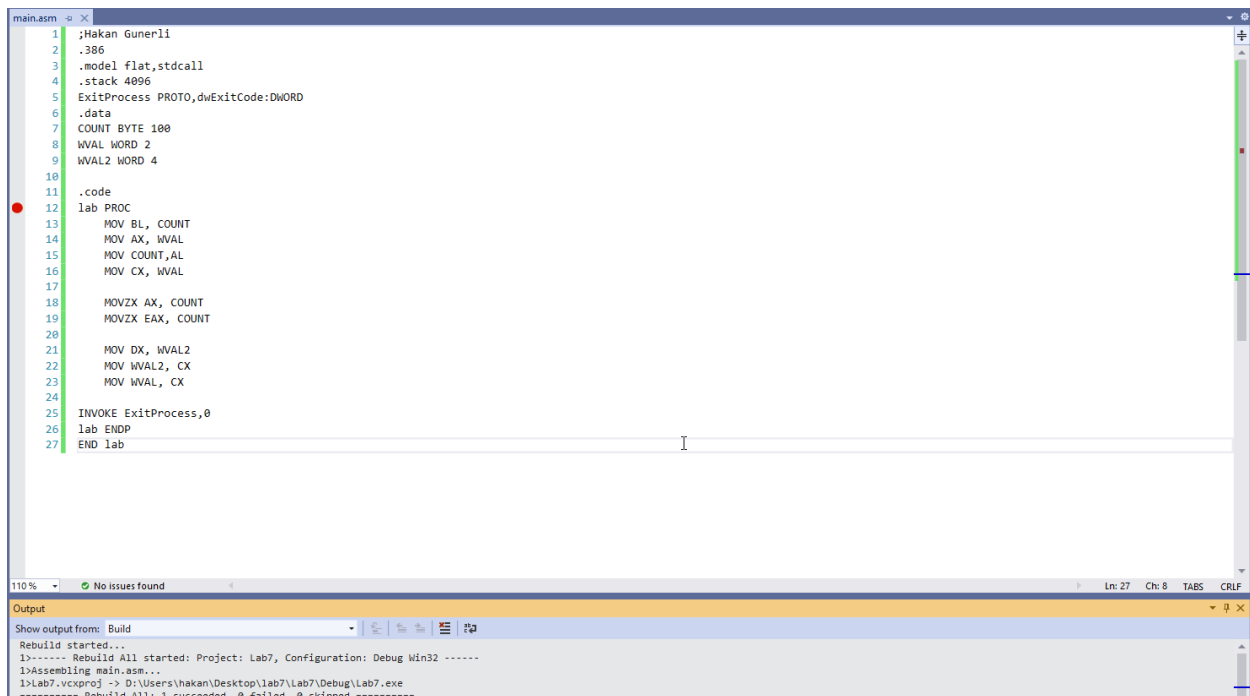


CSC 3210  
Computer Organization and Programming  
Lab 7  
Answer Sheet

Student Name: Hakan Gunerli  
Section: 014

Lab 7(a)

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



```
1 ;Hakan Gunerli
2 .386
3 .model flat,stdcall
4 .stack 4096
5 ExitProcess PROTO,dwExitCode:DWORD
6 .data
7 COUNT BYTE 100
8 WVAL WORD 2
9 WVAL2 WORD 4
10
11 .code
12 lab PROC
13     MOV BL, COUNT
14     MOV AX, WVAL
15     MOV COUNT,AL
16     MOV CX, WVAL
17
18     MOVZX AX, COUNT
19     MOVZX EAX, COUNT
20
21     MOV DX, WVAL2
22     MOV WVAL2, CX
23     MOV WVAL, CX
24
25     INVOKE ExitProcess,0
26 lab ENDP
27 END lab
```

Output

```
Show output from: Build
Rebuild started...
1>----- Rebuild All started: Project: Lab7, Configuration: Debug Win32 -----
1>Assembling main.asm...
1>Lab7.vcxproj -> D:\Users\hakan\Desktop\lab7\Lab7\Debug\Lab7.exe
***** Rebuild All: 1 succeeded, 0 failed, 0 skipped *****
```

## Lab 7(b)

Debug through each line of instructions.

Take screenshot that includes code and register window.

Record the register content.

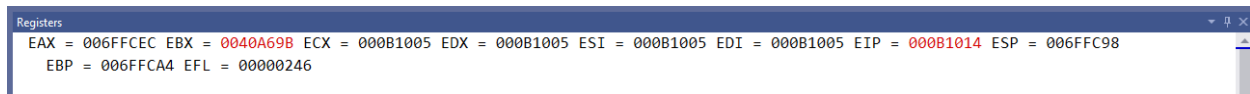
and explain the register contents.

Line number: 9

Instruction: mov bx, 0A69Bh

Register values: EAX = GARBAGE, EBX = A69Bh

Screenshot:



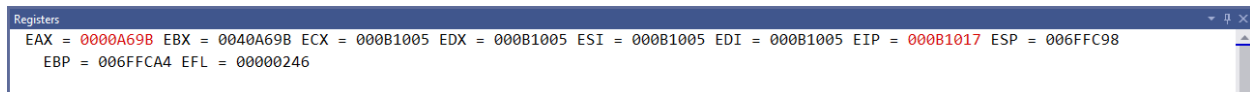
Explanation: All registers except bx are garbage, 0A69Bh moved to bx.

Line number: 10

Instruction: movzx eax, bx

Register values: EAX = 0000A69Bh, EBX = A69Bh

Screenshot:



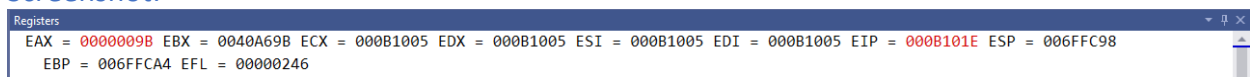
Explanation: bx value has been moved to eax. The value in ebx and bx stays the same.

Line number: 11

Instruction: movzx eax, myByte1

Register values: EAX = 0000009Bh, EBX = \_\_\_\_ A69Bh

Screenshot:



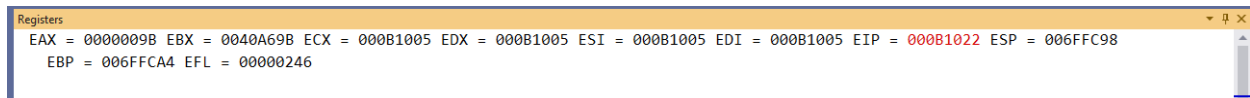
Explanation: The value in myByte1 (9Bh) moved to EAX.

Line number: 12

Instruction: mov bx, 0A69Bh

Register values: EAX = 0000009Bh, EBX = A69Bh

Screenshot:



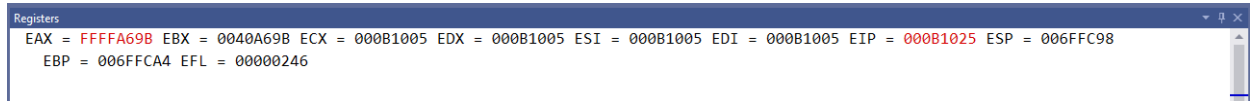
Explanation: The value 0A69Bh moved to the bx register, even though the value already exists in it. There is no difference in values of any register.

Line number: 13

Instruction: movsx eax, myByte1

Register values: EAX = FFFFA69Bh, EBX = \_\_\_\_A69Bh

Screenshot:



Explanation: value in the bx registers moved to eax after it got flashed with Fs (it is the signed).

Lab 7(c)

Debug through each line of instructions.

Take screenshot that includes code and register window.

Record the register content.

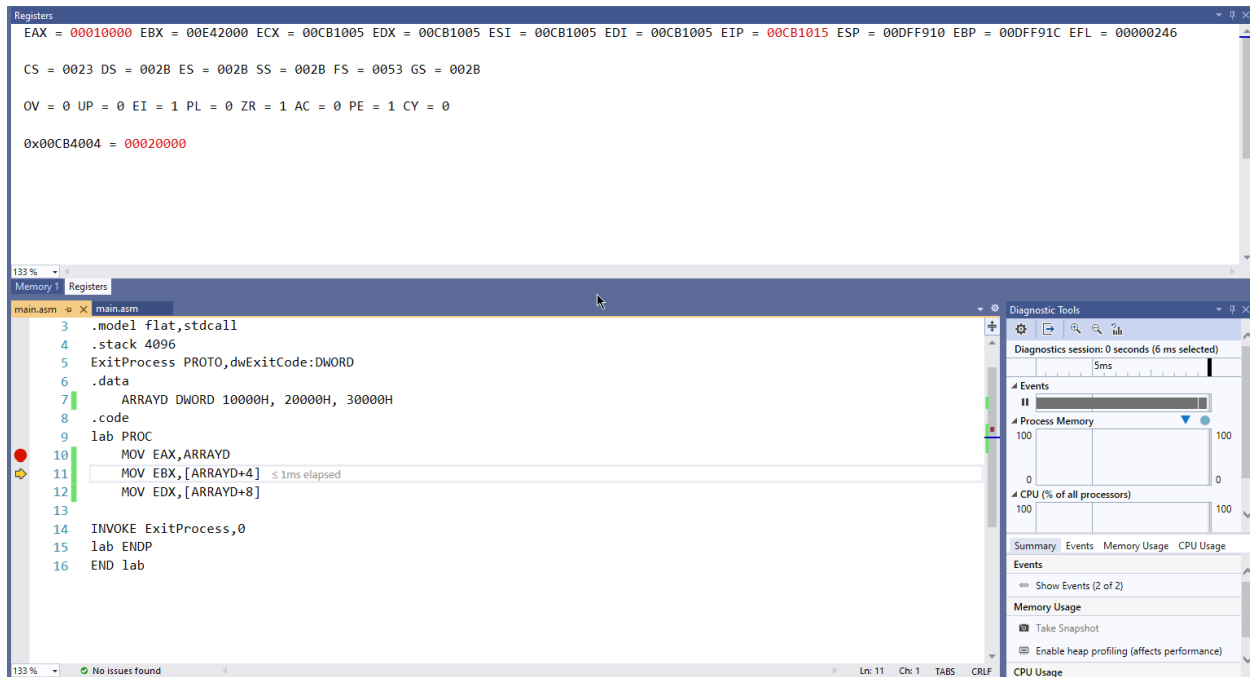
and explain the register contents.

Line number: 10

Instruction: mov eax, arrayd

Register values: eax= 0010000h

Screenshot:



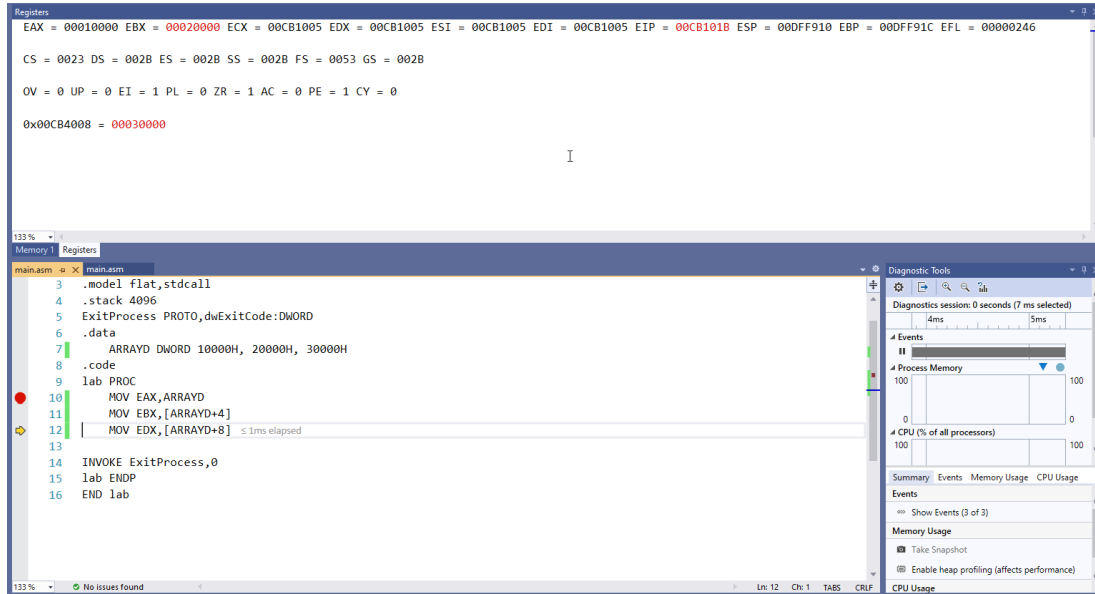
Explanation: first element is moved to eax.

Line number: 11

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

Register values: EAX = 0010000h, EBX = 0020000h

Screenshot:



Explanation:

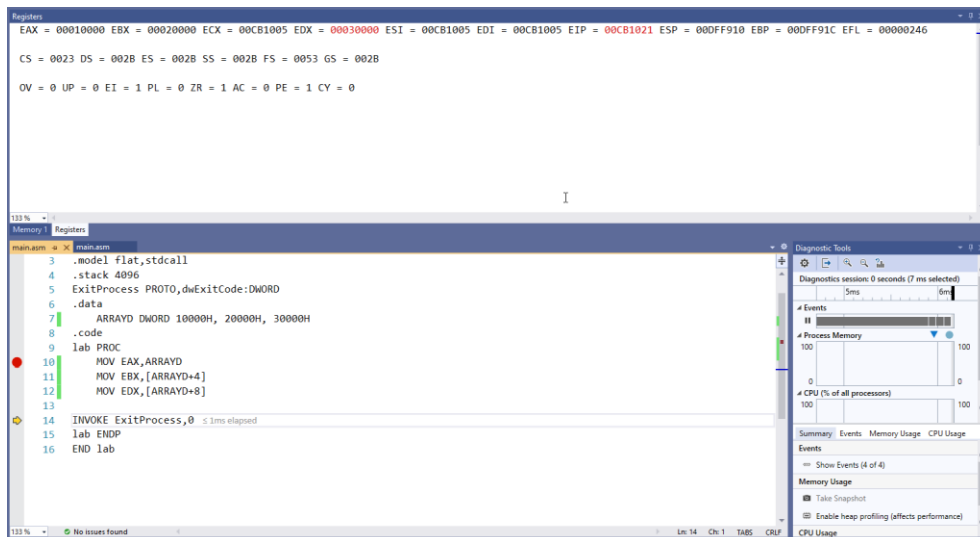
Second element is moved to ebx. Ebx = 0020000h

Line number: 12

Instruction: `mov edx [arrayD+8]`

Register values: EAX = 0010000h, EBX = 0020000h, EDX = 0030000h

Screenshot:



Explanation:

Third element of the array is moved to edx, edx contains 0030000h.

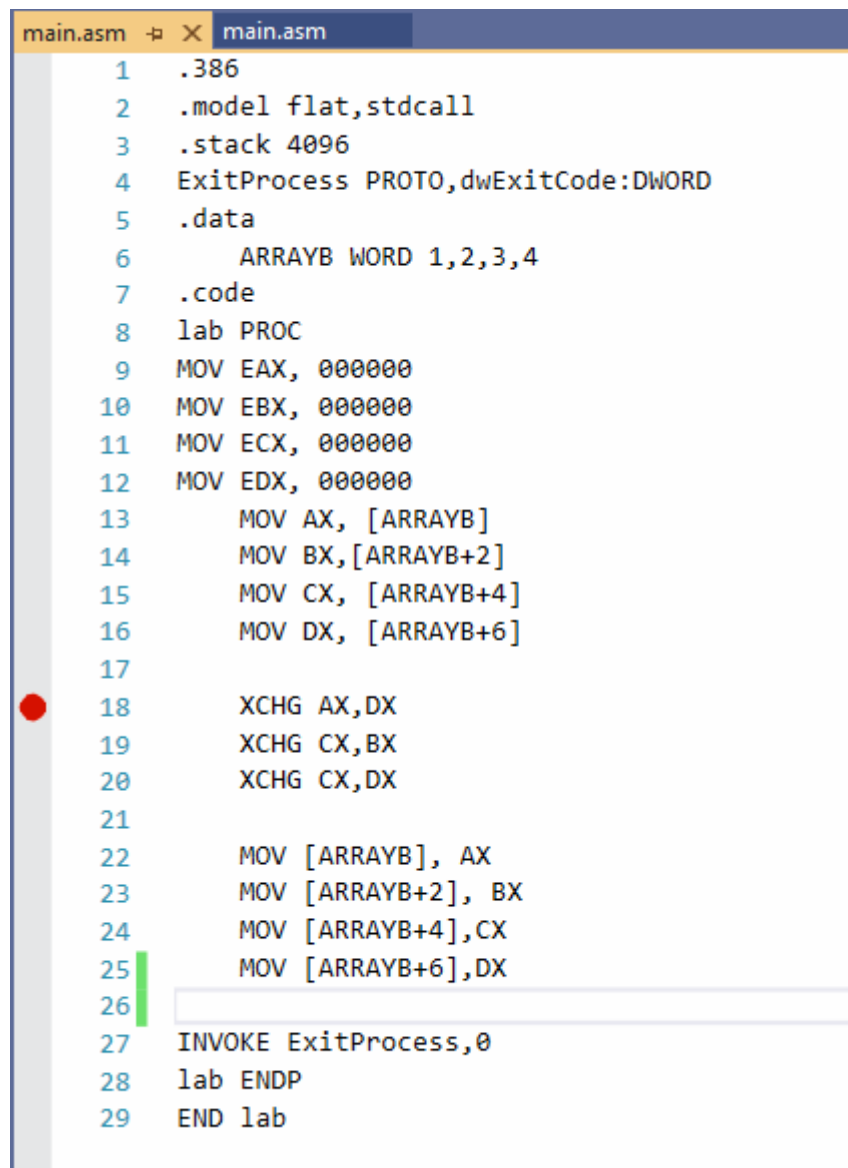
### Lab 7(d)

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 image shows a screenshot of an assembly code editor with a tab labeled 'main.asm'. The code is written in MASM syntax. It starts with a 386 instruction, sets the model to flat and stdcall, and reserves 4096 bytes of stack. It defines a procedure 'lab' that takes no arguments. Inside the procedure, it initializes EAX, EBX, ECX, and EDX to 0. It then loads the first four words of the 'ARRAYB' array into AX, BX, CX, and DX. Next, it performs three XCHG instructions: XCHG AX, DX; XCHG CX, BX; and XCHG CX, DX. Finally, it stores the modified values back into the array: MOV [ARRAYB], AX; MOV [ARRAYB+2], BX; MOV [ARRAYB+4], CX; and MOV [ARRAYB+6], DX. The procedure ends with INVOKE ExitProcess, 0, and the code ends with END lab.

```
1 .386
2 .model flat,stdcall
3 .stack 4096
4 ExitProcess PROTO,dwExitCode:DWORD
5 .data
6     ARRAYB WORD 1,2,3,4
7 .code
8 lab PROC
9     MOV EAX, 000000
10    MOV EBX, 000000
11    MOV ECX, 000000
12    MOV EDX, 000000
13    MOV AX, [ARRAYB]
14    MOV BX, [ARRAYB+2]
15    MOV CX, [ARRAYB+4]
16    MOV DX, [ARRAYB+6]
17
18    XCHG AX,DX
19    XCHG CX,BX
20    XCHG CX,DX
21
22    MOV [ARRAYB], AX
23    MOV [ARRAYB+2], BX
24    MOV [ARRAYB+4], CX
25    MOV [ARRAYB+6], DX
26
27    INVOKE ExitProcess,0
28 lab ENDP
29 END lab
```

### 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.

The screenshot shows a debugger window with two main panes. The top pane, titled 'Registers', displays the state of various CPU registers. The bottom pane, titled 'main.asm', shows the assembly code being debugged. A red dot on the left margin of the assembly pane indicates the current instruction pointer.

**Registers:**

Register	Value
EAX	0000000C
EBX	0000FFD2
ECX	00DF1005
EDX	0000FFD0
ESI	00DF1005
EDI	00DF1005
EIP	00DF1035
ESP	005CFDE8
EBP	005CFDF4
EFL	00000216

**main.asm:**

```
1 ;Hakan Gunerli
2 .386
3 .model flat,stdcall
4 .stack 4096
5 ExitProcess PROTO,dwExitCode:DWORD
6 .data
7     VAL1 SWORD 23
8     VAL2 SWORD -35
9     VAL3 SDWORD 4
10
11
12 .code
13 lab PROC
14     MOV EAX, VAL3
15
16     ADD EAX, VAL3
17     ADD EAX, VAL3 ; MULTIPLY BY 3
18
19     MOVZX EBX, VAL1
20     MOVZX EDX, VAL2
21
22     NEG EBX
23     ADD EBX, EDX
24     ADD EBX, EAX
25
26
27 INVOKE ExitProcess,0 < 1ms elapsed
28 lab ENDP
29 END lab
```