CSC 3210

Computer Organization and Programming

Lab 6

Answer Sheet

Student Name: **Hemant Kosaraju**

Section: **020**

Debug through each line of code and explain the register content

Lab 6(a)

Line number: **14**

Instruction: **mov bx, 0A69Bh**

Register value: **EBX = 00A5A69B**

Explanation: **The EBX register value is created or set to be any four values at the beginning #### however the last 4 digits are A69B because bx is the name for the 16-bit register value of EBX and each value for hexadecimal is 4 bits so 4 bits \* 4 = 16 bits which is equal to bx register = ####A69B**

**A screenshot of a computer

Description automatically generated**

Line number: **15**

Instruction: **movzx eax, bx**

Register value: **EAX = 0000A69B**

Explanation: **The EAX register value is created or set to be currently 0000A69B and this is due to there being a movzx instruction which is a zero extension instruction of the mov instruction and any values which are not assigned originally are assigned with value of 0 which can explain why we see EAX = 0000A69B**

A screenshot of a computer

Description automatically generated

Line number: **17**

Instruction: **movzx eax, myByte1**

Register value: **EAX = 0000009B**

Explanation: **The EAX register value is created or set to be currently 0000009B and this is due to the same movzx instruction which assigns the value 0 to unintialized or undeclared values from the previous instruction and puts the variable value of myByte1 which is 9Bh at the ending two digits of the EAX register**

A screenshot of a computer

Description automatically generated

Line number: **19**

Instruction: **mov bx, 0A69Bh**

Register value: **No register change observed; however in the background theoretically and logically actually the EBX register’s last 4 digits do update to A69B in the background though we just do not see it until it is set to the eax register using the sign extension**

Explanation: **There is no change though the ebx register’s 16-bit which is the bx register changed its last 4 digits to A69B; however only when line 20 is executed that is when we can see the 0000 values at the beginning becoming signed causing them to be FFFF and the last 4 digits become A69B therefore, in the next instruction line the output should be FFFFA69B**

A screenshot of a computer

Description automatically generated

Line number: **20**

Instruction: **movsx eax, bx**

Register value: **EAX = FFFFA69B, EIP = 00E41025**

Explanation: **This instruction of movsx changed the register value EAX from 0000A69B to FFFFA69Bh because the movsx instruction is another instruction with a similar function of zx except instead of being a zero extension the sx signed extension will flip the 0000 or uninitialized bits to FFFF, the movsx is the mov instruction with signed extension; both the movsx and movzx are important helpers if you are looking to mov information with different memory declarations such as a 16-bit register and an 8-bit value to allow it to function without returning an error**

A screenshot of a computer

Description automatically generated

Lab 6(b)

Line number: **16**

Instruction: **mov eax, arrayD**

Register value: **EAX = 00010000, the memory address of &arrayD = 00020000**

Explanation: **The EAX value is assigned to the first word value of arrayD which is 10000 and the memory address of &arrayD will show the next word value of arrayD which is 20000**

A screenshot of a computer

Description automatically generated

Line number: **17**

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

Register value: **EBX = 00020000, the memory address of &arrayD = 00030000**

Explanation: **The EBX register value is assigned to the fourth memory value of arrayD which would be 20000 and the memory address of &arrayD will show the next word value of arrayD which is 30000, this is because when we utilized word on the previous lab we saw that each word was 2 Bytes however we are now assigning the variable arrayD with 4 Bytes because WORD is 16 bit and 1 Byte is 8 -bits therefore 2 Bytes is 16-bits. DWORD is 32-bit therefore using the same logic it would be 4 Bytes; 32-bit. The second DWORD is 20000h which is why EAX = 00020000**

A screenshot of a computer

Description automatically generated

Line number: **18**

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

Register value: **EDX = 00030000**

Explanation: **The EDX register value is 00030000 because the arrayD variable is of DWORD type and 10000h would be 0 Bytes, 20000h would be 4 Bytes, and 30000h would be 8 Bytes therefore when the instruction mov edx, [arrayD + 8] is run the EDX register value would get the word at 8 Bytes in arrayD variable, which would be the 30000h value that is why we see EDX register is 00030000**

A screenshot of a computer

Description automatically generated

Lab 6(c)

A computer screen shot of a black screen

Description automatically generated

(copy the above format if you need more )