CSC 3210

Computer Organization and Programming

Lab 7

Answer Sheet

Student Name: **Hemant Kosaraju**

Section: **020**

Lab 7(a)

* 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 = (-Val3 - val2) + (val1\*2)

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

* Build and run the program using the debugger
* Examine the content of the registers

Take ONE screenshot showing your codes and the final result in EBX (EBX = 0000004D)

A computer screen shot of a black screen

Description automatically generated

Lab 7(b)

Debug through each line of instructions.

Record the register content.

and explain the register contents.

Line number: **17**

Instruction: mov ax, [esi] ; a. AX = **2010**

Register values: **AX = 2010**

Explanation: **The AX register has a register value of 16-bit and the previous line had the instruction mov esi, OFFSET myBytes which is BYTE type 8-bit and therefore since AX can hold two 8-bit values the 10h and 20h, therefore AX would be assigned with the 20h and 10h making the register value 2010**

Line number: **18**

Instruction: mov eax, DWORD PTR myWords ; b. EAX = **003B008A**

Register values: **EAX = 003B008A**

Explanation: **The instruction shows that a DWORD value will be moved to eax register however the DWORD will use PTR [Part or Partition] Operator, this operator will get the first two values of the myWords WORD 16-bit variable which are 8Ah and 3Bh, then the DWORD is 32-bit so to fill the remaining parts of the register it will assign zero to empty places therefore final result is accurate by showing 003B – first 16-bit 008A – second 16-bit which will be EAX = 003B008A**

Line number: **20**

Instruction: mov ax, [esi+2] ; c. AX = **003B**

Register values: **AX = 003B**

Explanation: **The previous line said mov esi, OFFSET myWords, because of this the memory OFFSET will contain the values at the specific location in the memory of the myWords variable, therefore mov ax, [esi+2] would get the second word in myWords variable because myWords variable is WORD type 16-bit and therefore each word is 2 Bytes then 0 Bytes is 8Ah and 2 Bytes would be 3Bh and that shows mov ax, [esi+2] would assign the value 003B to last 4 16-bit digits of EAX register or in other words the AX register**

Line number: **21**

Instruction: mov ax, [esi+6] ; d. AX = **0044**

Register values: **AX = 0044**

Explanation: **The instruction mov ax, [esi+6] will get the 6 [sixth] Byte of the myWords variable which would be 6 [sixth] Byte / 2 Bytes = 3 [Third] word which is 44h since the words start from 0, 1, 2, 3 the third word being 44h and then that index will be moved to the AX 16-bit register and is the explanation for the value showing AX = 0044**

Line number: **22**

Instruction: mov ax, [esi-4] ; e. AX =**2010**

Register values: **AX = 2010**

Explanation: **The** **mov ax, [esi-4] would be indexing outside the range of the myWords variable and because of this it might contain the previous values from the myBytes variable which can explain why the 10h was added to the AX register and the sign extension could have caused 20h to be added to the AX register, the result of the AX register would be 2010**

Take ONE screenshot that showing your codes in the debugging mode with register values

A screenshot of a computer

Description automatically generated

Lab 7(c)

Line number: **16**

Instruction: mov ax, WORD PTR [varB+2]

AX Register values: **AX = 0502**

Explanation: **The instruction mov ax, WORD PTR [varB + 2] would be using the PTR [Part or Partition] operator to get the word at array location [varB + 2] which would be the third Byte in the variable 02h and because the WORD type is 16-bit and the varB is BYTE type 8-bit it will get the 02h array value and the next 8-bit value which is 05h to be able to complete the 16-bit WORD data type and AX 16-bit register which would explain the result AX = 0502 register value**

Line number: **17**

Instruction: mov bl, BYTE PTR varD

BL Register values: **BL = 78**

Explanation: **The BL register value is 78 which can be explained because using the PTR [Part or Partition] operator the variable varD is a DWORD type 32-bit whereas the data type we are calculating the result to, would be, BYTE 8-bit and therefore the PTR operator would get the last two 8-bit digits from the varD variable which is 78 and that would set to being the BL 8-bit register**

Line number: **18**

BL Instruction: mov bl, BYTE PTR [varW+2]

Register values: **BL = 02**

Explanation: **The BL register value is 02 which can be explained by understanding that the varW variable has a data type WORD which is 16-bit and we are calculating the result to BYTE 8-bit, each word in the varW would be 2 Bytes because the data type is WORD 16-bit and therefore [varW + 2] will be 1202h and from here the BYTE PTR [Part or Partition] operator would just get the 8-bit value 02h from the 16-bit value 1202h which is then set to the BL register**

Line number: **19**

Instruction: mov ax, WORD PTR [varD+2]

AX Register values: **AX = 1234**

Explanation: **The AX register value is 1234 because when the instruction mov ax, WORD PTR [varD + 2] is run, the varD variable is DWORD 32-bit and therefore if a BYTE is 1 byte and 8-bit data type then each word of the DWORD data type should be 32-bit / 8-bit [1 BYTE] = 4 BYTES and therefore [varD + 2] would get not the whole word though rather just the first half of the varD array value which is 1234h and then since it is already in WORD 16-bit type now the AX register = 1234**

Line number: **20**

Instruction: mov eax, DWORD PTR varW

EAX Register values: **EAX = 12026543**

Explanation: **The value for the EAX register is 12026543 and this is explained through seeing that varW is a WORD 16-bit data type and just getting the varW variable should return 6543h however because we are going from WORD 16-bit to DWORD 32-bit this is a complete register and there will be space remaining, therefore to set the 32-bit eax register the PTR [Part or Partition] operator would also pick up the value at the next two bytes of varW which is 1202h and therefore it sets the EAX register to equal = 12026543**

Take ONE screenshot that showing your codes in the debugging mode with register values

A computer screen shot of a black screen

Description automatically generated

* **Problem LAB 7(d):** Write and run a program to find the values of a memory location and a register:

.data

dVal DWORD ?

.code

mov dVal,12345678h

mov ax,WORD PTR dVal+2

add ax,3

mov WORD PTR dVal,ax ; dVal= **12341237**

mov eax,dVal ;EAX= **12341237**

Line number: **18**

Instruction: mov WORD PTR dVal, ax

***Memory dVal*** values: **dVal = 12341237**

Explanation: **The explanation of the dVal memory value being 12341237 after typing &dVal in the address search box of the memory window I saw that the initial value of dVal was moved to be 12345678h however after debugging the instruction line mov WORD PTR dVal, ax it saved the first 16-bit values which were 1234 and then moved the 16-bit ax register values which were 1237 to the memory value of the dVal variable causing it to become 12341237 for the memory value at &dVal or 0x00E34000**

Line number: **19**

Instruction: mov eax,dVal

***EAX Register*** values: **EAX = 12341237**

Explanation: **The EAX register value is explained because the dVal variable was indexed using the part or partition PTR operator which got the first 16-bit value and the other ending 16-bit value was received from the ax register then the complete total value was stored as a 32-bit DWORD value in the memory address of the dVal variable, this line of instruction mov eax, dVal just took the value assigned to dVal and sent it to be displayed on the register output for eax using the mov data transfer instruction**

Take ONE screenshot that showing your codes in the debugging mode with register values & the dVal variable in memory windows

A computer screen shot of a black screen

Description automatically generated

**Problem LAB 7(e):**

Line number: **16**

Instruction: mov eax, TYPE myBytes ; a.

EAX Register values: **EAX = 00000001**

Explanation: **This is because the TYPE operator represents the size in bytes of a data type and because the data type of myBytes variable was BYTE, the instruction mov eax, TYPE myBytes would return 1 because each word in a BYTE Data Type is 1 Byte**

Line number: **17**

Instruction: mov eax, LENGTHOF myBytes ; b.

eax Register values: **EAX = 00000004**

Explanation: **The EAX register for the instruction mov eax, LENGTHOF myBytes is 00000004 because the LENGTHOF operator counts the number of elements included in the data type variable and for myBytes BYTE data type we have 10h, 20h, 30h, 40h which shows that 10h would be 1 20h would be 2 30h would be 3 and 40h would be 4**

Line number: **18**

Instruction: mov eax, SIZEOF myBytes ; c.

eax Register values: **EAX = 00000004**

Explanation: **The EAX register for the instruction mov eax, SIZEOF myBytes did not change the register value of EAX and was still 00000004 which is because SIZEOF returns a value that is the value of the LENGTHOF multiplied by the TYPE which would be LENGTHOF 4 elements × TYPE 1 BYTE = 00000004, which explains no change in the EAX register**

Line number: **19**

Instruction: mov eax, TYPE myWords

eax Register values: **EAX = 00000002**

Explanation: **The EAX register for this instruction mov eax, TYPE myWords is 00000002 because the variable myWords uses data type WORD 16-bit which has a byte value of 2 bytes / [per] word that explains EAX being 00000002**

Line number: **20**

Instruction: mov eax, LENGTHOF myWords

eax Register values: **EAX =** **00000004**

Explanation: **This is because the instructions mov eax, LENGTHOF myWords counts how many elements are in the variable declaration of myWords which is of WORD 16-bit data type and because the myWords variable consisted of 3 DUP (?), 2000h that would mean 1- ?, 2-?, 3-?, and 4 being 2000h which is why EAX = 00000004 for the LENGTHOF myWords variable**

Line number: **21**

Instruction: mov eax, SIZEOF myWords

eax Register values: **EAX = 00000008**

Explanation: **The EAX register value is now set to 00000008 after debugging the instruction mov eax, SIZEOF myWords because the TYPE was 2 Bytes and the LENGTHOF myWords was 4 elements and LENGTHOF 4 elements × TYPE 2 Bytes = EAX register value 00000008, this is because SIZEOF is said to be the LENGTHOF the data variable × [multiplied by] the TYPE of the data Type [in Bytes – based on if the Data Type is 8-bit, 16-bit, 32-bit, or even 64-bit and higher]**

Line number: **22**

Instruction: mov eax, SIZEOF myString

eax Register values: **EAX = 00000005**

Explanation: **The EAX register value after executing the instruction line mov eax, SIZEOF myString would be 00000005 and how we can understand that is because myString is of Data Type BYTE which is TYPE 1 BYTE and the LENGTHOF even though the string is “ABCDE” the compiler or file interprets them as single characters with 1 byte each therefore LENGTHOF would be 5 elements and LENGTHOF 5 elements** × **TYPE 1 BYTE would make the SIZEOF myString variable be 5 which is set to the EAX register = 00000005**

Take ONE screenshot that showing your codes in the debugging mode with register values

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated